ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 316 (2004) 153–190

Theoretical Computer Science

www.elsevier.com/locate/tcs

Domain theory for concurrency

Mikkel Nygaard[a,*,1], Glynn Winskel[b]

[a] *Basic Research in Computer Science (BRICS), University of Aarhus, Denmark*
[b] *Computer Laboratory, University of Cambridge, Cambridge, UK*

Abstract

A simple domain theory for concurrency is presented. Based on a categorical model of linear logic and associated comonads, it highlights the role of linearity in concurrent computation. Two choices of comonad yield two expressive metalanguages for higher-order processes, both arising from canonical constructions in the model. Their denotational semantics are fully abstract with respect to contextual equivalence. One language derives from an exponential of linear logic; it supports a straightforward operational semantics with simple proofs of soundness and adequacy. The other choice of comonad yields a model of affine-linear logic, and a process language with a tensor operation to be understood as a parallel composition of independent processes. The domain theory can be generalised to presheaf models, providing a more refined treatment of nondeterministic branching. The article concludes with a discussion of a broader programme of research, towards a fully fledged domain theory for concurrency.

Keywords: Domain theory; Concurrency; Path semantics; Presheaf semantics

1. Introduction

Denotational semantics and domain theory of Scott and Strachey provide a *global* mathematical setting for sequential computation, and thereby place programming languages in connection with each other; connect with the mathematical worlds of algebra, topology and logic; and inspire programming languages, type disciplines and methods of reasoning.

In concurrent/distributed/interactive computation that global mathematical guidance is missing, and domain theory has had little direct influence on theories of concurrent

* Corresponding author.
E-mail address: nygaard@daimi.au.dk (M. Nygaard).
URL: www.brics.dk
[1] Funded by the Danish National Research Foundation.

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.01.029

computation. One reason is that classical domain theory has not scaled up to the more intricate models used there.

Broadly speaking, approaches to concurrency are either based on a specific mathematical model of processes or start from the syntax of a process calculus. Among the variety of models for concurrency, one can discern an increasing use of causal/independence/partial-order models (such as Petri nets and event structures) in which computation paths are partial orders of events. Independence models thread through partial-order model checking [45], security protocols [50], nondeterministic dataflow [17], self-timed circuits [18], term-rewriting, game semantics [3], and the analysis of distributed algorithms [29]. There are a variety of process calculi, most of them based on an operational semantics. Following on from the π-calculus [36,47], new-name generation is central to almost all calculi of topical interest. Many are higher-order (allowing process passing) which presents a challenge in understanding suitable equivalences, of which forms of bisimulation are prevalent.

Theories of concurrency form a rather fragmented picture. Relations between different approaches are often unclear; ideas are rediscovered (for example, special event structures reappear as "strand spaces" in reasoning about security protocols [16,50]). A lot of energy is used on local optimisations to specific process calculi, optimisations that may obscure connections and the global picture. Research is often "modelling-driven" in the sense that many approaches are based on formalising some feature observed in the computing world; the feature may be general such as locality of computation, or specific as in the study of a particular protocol. But the lessons learnt often remain isolated for lack of the commonality a global framework would provide.

A domain theory which handled higher-order processes, independence models, name-generation, and possessed an operational interpretation would provide a global mathematical framework for most theories of concurrency. In case incorporating independence models into a domain theory seems a tall order, there are now arguments (based on event-structure representations of process denotations—see Section 6.4) that the operational semantics associated with a domain theory for concurrency will involve event structures. It should be remarked that a *traditional* use of powerdomains [46], based on domains of resumptions, will fall short because, insisting on a nondeterministic choice of actions one at a time, it cannot accommodate independence models where computation paths have more structure than strings of actions.

How do we work towards such a domain theory for concurrency? The potentially complicated structure of computation paths suggests building a domain theory directly on computation paths. This line has been followed in what seemed originally to be two different directions, one being Matthew Hennessy's semantics for CCS with process passing [20], in which a process denotes the set of its computation paths. We will call this kind of semantics a *path semantics* because of its similarity to trace semantics [24]; in both cases, processes denote downwards-closed sets of computation paths and the corresponding notion of process equivalence, called *path equivalence*, is given by equality of such sets. Computation paths, however, may have more structure than traditional traces, e.g., allowing path semantics to take nondeterministic branching into account in a limited way. For example, path equivalence is related to simulation equivalence in Section 3.5. The other path-based approach is that of categories of

presheaf models [12] in which processes denote mappings from computation paths to sets of "realisers" saying *how* each computation path may be realised. This extra structure allows the incorporation of complete branching information, and the corresponding notion of process equivalence is a form of bisimulation [26]. The two approaches are variations on a common idea: that a process denotes a form of characteristic function in which the truth values are sets of realisers. A path set may be viewed as a special presheaf that yields at most one realiser for each path.

The study of presheaf models for concurrency has drawn attention to a 2-categorical model of linear logic and associated pseudo-comonads [15]. This led to the discovery of two expressive metalanguages for concurrency, one based on an exponential of linear logic (from which one derives a model of intuitionistic logic), the other based on a weakening comonad (from which one derives a model of affine-linear logic). The presheaf semantics led to operational semantics, guided by the idea that derivations of transitions in the operational semantics, associated with paths, should correspond to elements of the presheaf denotations. The presheaf models capture the nondeterministic branching of processes and support notions of bisimulation. But there is a significant overhead in terms of the category theory needed.

In this paper, we concentrate on the simpler path semantics of the languages. Path sets give rise to a simpler version of the categorical models, avoiding the 2-categorical structure. Though path sets are considerably simpler than presheaves they furnish models which are sufficiently rich in structure to show how both languages arise from canonical constructions on path sets. The path semantics admits simple proofs of full abstraction, showing that path equivalence coincides with contextual equivalence.

One language, called HOPLA for higher-order process language [42,43], derives from an exponential of linear logic. It can be viewed as an extension of the lambda calculus with CCS-like nondeterministic sum and prefix operations, in which types express the form of computation path of which a process is capable. HOPLA can directly encode calculi like CCS [35], CCS with process passing [22], and mobile ambients with public names [10,11], and it can be given a straightforward operational semantics supporting a standard bisimulation congruence. We relate the denotational and operational semantics giving pleasingly simple proofs of soundness and adequacy. Full abstraction implies that contextual equivalence coincides with logical equivalence for a fragment of Hennessy–Milner logic, linking up with simulation equivalence [21]. Work is in progress on extending HOPLA with name generation [54].

The other language is here called Affine HOPLA [41] and is based on a weakening comonad that yields a model of affine-linear logic in the sense of Jacobs [25]. This language adds to HOPLA an interesting tensor operation at the price of linearity constraints on the occurrences of variables. The tensor can be understood as a parallel composition of independent processes and allows Affine HOPLA to encode processes of the kind found in treatments of nondeterministic dataflow [27].

We conclude with a discussion of how the results fit within a broader programme of research, towards a fully fledged domain theory for concurrency. Important leads come by moving to categories obtained from presheaves rather than path sets. These categories are very rich in structure. They point towards more expressive languages than HOPLA and Affine HOPLA. In particular, the affine category accommodates the

independence model of event structures to the extent of supporting the standard event structure semantics of CCS and related languages [14], as well as the trace of nondeterministic dataflow [23]. In fact, Affine HOPLA can be given an event structure semantics which at first order provides a representation of the presheaf denotations. Nevertheless, it is here we meet the limitations of Affine HOPLA, and HOPLA. They can be shown not to support definitions of the standard event structure semantics of CCS and the trace of nondeterministic dataflow [40].

2. Domain theory of path sets

In the path semantics, processes are intuitively represented as collections of their computation paths. Paths are elements of preorders $\mathbb{P}, \mathbb{Q}, \ldots$ called *path orders* which function as process types, each describing the set of possible paths for processes of that type together with their sub-path ordering.[2] A process of type $\mathbb{P}$ is then represented as a downwards-closed subset $X \subseteq \mathbb{P}$, called a *path set*. Path sets ordered by inclusion form the elements of the poset $\widehat{\mathbb{P}}$ which we will think of as a domain of meanings of processes of type $\mathbb{P}$.

The poset $\widehat{\mathbb{P}}$ has many interesting properties. First of all, it is a complete lattice with joins given by union. In the sense of Hennessy and Plotkin [22], $\widehat{\mathbb{P}}$ is a "nondeterministic domain", with joins used to interpret nondeterministic sums of processes. Accordingly, given a family $(X_i)_{i\in I}$ of elements of $\widehat{\mathbb{P}}$, we will often write $\sum_{i\in I} X_i$ for their join. A typical finite join is written $X_1 + \cdots + X_k$ while the empty join is the empty path set, the inactive process, written $\varnothing$.

A second important property of $\widehat{\mathbb{P}}$ is that any $X \in \widehat{\mathbb{P}}$ is the join of certain "prime" elements below it; $\widehat{\mathbb{P}}$ is a *prime algebraic complete lattice* [39]. Primes are down-closures $\mathrm{y}_{\mathbb{P}}\, p = \{p' : p' \leqslant_{\mathbb{P}} p\}$ of individual elements $p \in \mathbb{P}$, representing a process that may perform the computation path p. The map $\mathrm{y}_{\mathbb{P}}$ reflects as well as preserves order, so that $p \leqslant_{\mathbb{P}} p'$ iff $\mathrm{y}_{\mathbb{P}}\, p \subseteq \mathrm{y}_{\mathbb{P}}\, p'$, and $\mathrm{y}_{\mathbb{P}}$ thus "embeds" $\mathbb{P}$ in $\widehat{\mathbb{P}}$. We clearly have $\mathrm{y}_{\mathbb{P}}\, p \subseteq X$ iff $p \in X$ and prime algebraicity of $\widehat{\mathbb{P}}$ amounts to saying that any $X \in \widehat{\mathbb{P}}$ is the union of its elements:

$$X = \bigcup_{p\in X} \mathrm{y}_{\mathbb{P}}\, p. \tag{1}$$

Finally, $\widehat{\mathbb{P}}$ is characterised abstractly as the *free join-completion* of $\mathbb{P}$, meaning (i) it is join-complete and (ii) given any join-complete poset C and a monotone map $f : \mathbb{P} \to C$, there is a unique join-preserving map $f^\dagger : \widehat{\mathbb{P}} \to C$ such that the diagram on the left below commutes.

$$\begin{array}{ccc} \mathbb{P} & \xrightarrow{\mathrm{y}_{\mathbb{P}}} & \widehat{\mathbb{P}} \\ & f \searrow \quad \downarrow f^\dagger & \\ & C & \end{array} \qquad f^\dagger X = \bigcup_{p\in X} fp. \tag{2}$$

We call $f^\dagger$ *the extension of* f *along* $\mathrm{y}_{\mathbb{P}}$. Uniqueness of $f^\dagger$ follows from (1).

[2] It is possible to work with straight posets rather than preorders—indeed, the mathematics is virtually unaffected by this choice—but preorders will be helpful in dealing with recursive types in Section 3.1.

Notice that we may instantiate C to any poset of the form $\widehat{\mathbb{Q}}$, drawing our attention to join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. By the freeness property (2), join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ are in bijective correspondence with monotone maps $\mathbb{P} \to \widehat{\mathbb{Q}}$. Each element Y of $\widehat{\mathbb{Q}}$ can be represented using its "characteristic function", a monotone map $f_Y : \mathbb{Q}^{\mathrm{op}} \to \mathbf{2}$ from the opposite order to the simple poset $0 < 1$ such that $Y = \{q : f_Y q = 1\}$ and $\widehat{\mathbb{Q}} \cong [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]$. Uncurrying then yields the following chain:

$$[\mathbb{P}, \widehat{\mathbb{Q}}] \cong [\mathbb{P}, [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]] \cong [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{2}] = [(\mathbb{P}^{\mathrm{op}} \times \mathbb{Q})^{\mathrm{op}}, \mathbf{2}] \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}} \tag{3}$$

So the order $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ provides a function space type. We will now investigate what additional type structure is at hand.

2.1. Linear and continuous categories

Write **Lin** for the category with path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. It turns out **Lin** has enough structure to be understood as a categorical model of Girard's linear logic [19,49]. Accordingly, we will call arrows of **Lin** *linear* maps.

Linear maps are represented by elements of $\widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$ and so by downwards-closed subsets of the order $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. This relational presentation exposes an involution central in understanding **Lin** as a categorical model of classical linear logic. The involution of linear logic, yielding $\mathbb{P}^{\perp}$ on an object $\mathbb{P}$, is given by $\mathbb{P}^{\mathrm{op}}$; clearly, downwards-closed subsets of $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ correspond to downwards-closed subsets of $(\mathbb{Q}^{\mathrm{op}})^{\mathrm{op}} \times \mathbb{P}^{\mathrm{op}}$, showing how maps $\mathbb{P} \to \mathbb{Q}$ correspond to maps $\mathbb{Q}^{\perp} \to \mathbb{P}^{\perp}$ in **Lin**. The tensor product of $\mathbb{P}$ and $\mathbb{Q}$ is given by the product of preorders $\mathbb{P} \times \mathbb{Q}$; the singleton order $\mathbb{1}$ is a unit for tensor. Linear function space $\mathbb{P} \multimap \mathbb{Q}$ is then obtained as $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. Products $\mathbb{P}$ & $\mathbb{Q}$ are given by $\mathbb{P} + \mathbb{Q}$, the disjoint juxtaposition of preorders. An element of $\widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$ can be identified with a pair (X, Y) with $X \in \widehat{\mathbb{P}}$ and $Y \in \widehat{\mathbb{Q}}$, which provides the projections $\pi_1 : \mathbb{P}$ & $\mathbb{Q} \to \mathbb{P}$ and $\pi_2 : \mathbb{P}$ & $\mathbb{Q} \to \mathbb{Q}$ in **Lin**. More general, not just binary, products $\&_{i \in I} \mathbb{P}_i$ with projections π_j, for $j \in I$, are defined similarly. From the universal property of products, a collection of maps $f_i : \mathbb{P} \to \mathbb{P}_i$, for $i \in I$, can be tupled together to form a unique map $\langle f_i \rangle_{i \in I} : \mathbb{P} \to \&_{i \in I} \mathbb{P}_i$ with the property that $\pi_j \circ \langle f_i \rangle_{i \in I} = f_j$ for all $j \in I$. The empty product is given by the empty order $\mathbb{0}$ and, as the terminal object, is associated with unique maps $\varnothing_{\mathbb{P}} : \mathbb{P} \to \mathbb{0}$, constantly $\varnothing$, for any path order $\mathbb{P}$. All told, **Lin** is a $*$-autonomous category, so a symmetric monoidal closed category with a dualising object, and has finite products (indeed, all products) as required by Seely's definition of a model of linear logic [49].

In fact, **Lin** also has all coproducts, also given on objects $\mathbb{P}$ and $\mathbb{Q}$ by the juxtaposition $\mathbb{P} + \mathbb{Q}$ and so coinciding with products. Injection maps $in_1 : \mathbb{P} \to \mathbb{P} + \mathbb{Q}$ and $in_2 : \mathbb{Q} \to \mathbb{P} + \mathbb{Q}$ in **Lin** derive from the obvious injections into the disjoint sum of preorders. The empty coproduct is the empty order $\mathbb{0}$ which is then a zero object. This collapse of products and coproducts highlights that **Lin** has arbitrary *biproducts*. Via the isomorphism $\mathbf{Lin}(\mathbb{P}, \mathbb{Q}) \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$, each homset of **Lin** can be seen as a commutative monoid with neutral element the always $\varnothing$ map, itself written $\varnothing : \mathbb{P} \to \mathbb{Q}$, and sum given by union, written $+$. Composition in **Lin** is bilinear in that, given $f, f' : \mathbb{P} \to \mathbb{Q}$

and $g, g' : \mathbb{Q} \to \mathbb{R}$, we have $(g+g') \circ (f+f') = g \circ f + g \circ f' + g' \circ f + g' \circ f'$. Further, given a family of objects $(\mathbb{P}_\alpha)_{\alpha \in A}$, we have for each $\beta \in A$ a diagram

$$\mathbb{P}_\beta \underset{in_\beta}{\overset{\pi_\beta}{\leftrightarrows}} \sum_{\alpha \in A} \mathbb{P}_\alpha \text{ such that } \begin{array}{r} \pi_\beta \circ in_\beta = 1_{\mathbb{P}_\beta}, \\ \pi_\beta \circ in_\alpha = \varnothing \text{ if } \alpha \neq \beta \\ \sum_{\alpha \in A} (in_\alpha \circ \pi_\alpha) = 1_{\sum_{\alpha \in A} \mathbb{P}_\alpha}. \end{array} \tag{4}$$

Processes of type $\sum_{\alpha \in A} \mathbb{P}_\alpha$ may intuitively perform computation paths in any of the component path orders $\mathbb{P}_\alpha$.

We see that **Lin** is rich in structure. But linear maps alone are too restrictive. Being join-preserving, they in particular preserve the empty join. So, unlike, e.g. prefixing, linear maps always send the inactive process $\varnothing$ to itself. Looking for a broader notion of maps between nondeterministic domains, we follow the discipline of linear logic and consider non-linear maps whose domain is under an exponential, !. One choice of a suitable exponential for **Lin** is got by taking $!\mathbb{P}$ to be the preorder obtained as the free finite-join completion of $\mathbb{P}$. Concretely, $!\mathbb{P}$ can be defined to have finite subsets of $\mathbb{P}$ as elements with ordering given by $\preccurlyeq_{\mathbb{P}}$, defined for arbitrary subsets X, Y of $\mathbb{P}$ as follows:

$$X \preccurlyeq_{\mathbb{P}} Y \Leftrightarrow_{\text{def}} \ \forall p \in X. \exists q \in Y.\ p \leqslant_{\mathbb{P}} q. \tag{5}$$

When $!\mathbb{P}$ is quotiented by the equivalence induced by the preorder we obtain a poset which is the free finite-join completion of $\mathbb{P}$. By further using the obvious inclusion of this completion into $\widehat{\mathbb{P}}$, we get a map $i_{\mathbb{P}} : !\mathbb{P} \to \widehat{\mathbb{P}}$ sending a finite set $\{p_1, \ldots, p_n\}$ to the join $\mathrm{y}_{\mathbb{P}} p_1 + \cdots + \mathrm{y}_{\mathbb{P}} p_n$. Such finite sums of primes are the finite (isolated, compact) elements of $\widehat{\mathbb{P}}$. The map $i_{\mathbb{P}}$ assumes the role of $\mathrm{y}_{\mathbb{P}}$ above. For any $X \in \widehat{\mathbb{P}}$ and $P \in !\mathbb{P}$, we have $i_{\mathbb{P}} P \subseteq X$ iff $P \preccurlyeq_{\mathbb{P}} X$, and X is the directed join of the finite elements below it:

$$X = \bigcup_{P \preccurlyeq_{\mathbb{P}} X} i_{\mathbb{P}} P. \tag{6}$$

Further, $\widehat{\mathbb{P}}$ is the *free directed-join completion* of $!\mathbb{P}$ (also known as the *ideal completion* of $!\mathbb{P}$). This means that given any monotone map $f : !\mathbb{P} \to C$ for some directed-join complete poset C, there is a unique directed-join preserving (i.e. Scott continuous) map $f^\ddagger : \widehat{\mathbb{P}} \to C$ such that the diagram below commutes.

$$\begin{array}{ccc} !\mathbb{P} & \xrightarrow{i_{\mathbb{P}}} & \widehat{\mathbb{P}} \\ & f \searrow \downarrow f^\ddagger & \\ & C & \end{array} \qquad f^\ddagger X = \bigcup_{P \preccurlyeq_{\mathbb{P}} X} fP. \tag{7}$$

Uniqueness of $f^\ddagger$, called the *extension of f along $i_{\mathbb{P}}$*, follows from (6). As before, we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (2) and (7), there is a bijective correspondence between linear maps $!\mathbb{P} \to \mathbb{Q}$ and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

We define the category **Cts** to have path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. These arrows allow more process operations, including prefixing, to be expressed. The structure of **Cts** is induced by that of **Lin** via an adjunction between the two categories.

2.2. An adjunction

As linear maps are continuous, **Cts** has **Lin** as a sub-category, one which shares the same objects. We saw above that there is a bijection

$$\mathbf{Lin}(!\mathbb{P}, \mathbb{Q}) \cong \mathbf{Cts}(\mathbb{P}, \mathbb{Q}). \tag{8}$$

This is in fact natural in $\mathbb{P}$ and $\mathbb{Q}$ so an adjunction with the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Cts}$ as right adjoint. Via (7) the map $\mathrm{y}_{!\mathbb{P}} : !\mathbb{P} \to \widehat{!\mathbb{P}}$ extends to a map $\eta_{\mathbb{P}} = \mathrm{y}^{\ddagger}_{!\mathbb{P}} : \mathbb{P} \to !\mathbb{P}$ in **Cts**. Conversely, $i_{\mathbb{P}} : !\mathbb{P} \to \widehat{\mathbb{P}}$ extends to a map $\varepsilon_{\mathbb{P}} = i^{\dagger}_{\mathbb{P}} : !\mathbb{P} \to \mathbb{P}$ in **Lin** using (2). These maps are the unit and counit, respectively, of the adjunction

$$\eta_{\mathbb{P}} X = \bigcup_{P \leqslant_{\mathbb{P}} X} \mathrm{y}_{!\mathbb{P}} P, \quad \varepsilon_{\mathbb{P}} X = \bigcup_{P \in X} i_{\mathbb{P}} P. \tag{9}$$

The left adjoint is the functor $! : \mathbf{Cts} \to \mathbf{Lin}$ given on arrows $f : \mathbb{P} \to \mathbb{Q}$ by $(\eta_{\mathbb{Q}} \circ f \circ i_{\mathbb{P}})^{\dagger} : !\mathbb{P} \to !\mathbb{Q}$. Bijection (8) then maps $g : !\mathbb{P} \to \mathbb{Q}$ in **Lin** to $\bar{g} = g \circ \eta_{\mathbb{P}} : \mathbb{P} \to \mathbb{Q}$ in **Cts** while its inverse maps $f : \mathbb{P} \to \mathbb{Q}$ in **Cts** to $\bar{f} = \varepsilon_{\mathbb{Q}} \circ !f$ in **Lin**. We call $\bar{g}$ and $\bar{f}$ the *transpose* of g and f, respectively; of course, transposing twice yields back the original map. As **Lin** is a sub-category of **Cts**, the counit is also a map in **Cts**. We have $\varepsilon_{\mathbb{P}} \circ \eta_{\mathbb{P}} = 1_{\mathbb{P}}$ and $1_{!\mathbb{P}} \leqslant \eta_{\mathbb{P}} \circ \varepsilon_{\mathbb{P}}$, the pointwise order, for all objects $\mathbb{P}$.

Right adjoints preserve products, and so **Cts** has finite products given as in **Lin**. Hence, **Cts** is a symmetric monoidal category like **Lin**, and in fact, our adjunction is symmetric monoidal (see [31, pp. 251–256]). In detail, there are isomorphisms of path orders

$$k : \mathbb{1} \cong !\mathbb{O} \quad \text{and} \quad m_{\mathbb{P},\mathbb{Q}} : !\mathbb{P} \times !\mathbb{Q} \cong !(\mathbb{P} \;\&\; \mathbb{Q}) \tag{10}$$

with $m_{\mathbb{P},\mathbb{Q}}$ mapping a pair $(P, Q) \in !\mathbb{P} \times !\mathbb{Q}$ to the union $in_1 P \cup in_2 Q$; any element of $!(\mathbb{P} \;\&\; \mathbb{Q})$ can be written on this form. These isomorphisms induce isomorphisms with the same names in **Lin** with m natural. Moreover, k and m commute with the associativity, symmetry and unit maps of **Lin** and **Cts**, such as $s^{\mathbf{Lin}}_{\mathbb{P},\mathbb{Q}} : \mathbb{P} \times \mathbb{Q} \cong \mathbb{Q} \times \mathbb{P}$ and $r^{\mathbf{Cts}}_{\mathbb{Q}} : \mathbb{Q} \;\&\; \mathbb{O} \cong \mathbb{Q}$, making ! symmetric monoidal. It then follows [28] that the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Cts}$ is symmetric monoidal as well, and that the unit and counit are monoidal transformations. Thus, there are maps

$$l : \mathbb{O} \to \mathbb{1} \quad \text{and} \quad n_{\mathbb{P},\mathbb{Q}} : \mathbb{P} \;\&\; \mathbb{Q} \to \mathbb{P} \times \mathbb{Q} \tag{11}$$

in **Cts**, with n natural, corresponding to k and m above; l maps $\varnothing$ to $\{*\}$ while $n_{\mathbb{P},\mathbb{Q}}$ is the extension $h^{\ddagger}$ of the map $h(in_1 P \cup in_2 Q) = i_{\mathbb{P}} P \times i_{\mathbb{Q}} Q$. The unit also makes the

diagrams below commute and the counit satisfies similar properties.

$$\begin{array}{ccccc} & & \mathbb{P}\&\mathbb{Q} & & \\ & \eta_{\mathbb{P}}\&\eta_{\mathbb{Q}}\swarrow & & \searrow \eta_{\mathbb{P}\&\mathbb{Q}} & \\ !\mathbb{P}\&!\mathbb{Q} & \xrightarrow[n_{!\mathbb{P},!\mathbb{Q}}]{} & !\mathbb{P}\times!\mathbb{Q} & \xrightarrow[n_{1\mathbb{P},\mathbb{Q}}]{} & !(\mathbb{P}\&\mathbb{Q}) \end{array} \qquad \begin{array}{ccc} \mathbb{O} & \xrightarrow{l} & \mathbb{1} \\ & \eta_{\mathbb{O}}\searrow & \downarrow k \\ & & !\mathbb{O} \end{array} \tag{12}$$

The diagram on the left can be written as $str_{\mathbb{P},\mathbb{Q}} \circ (1_{\mathbb{P}} \ \& \ \eta_{\mathbb{Q}}) = \eta_{\mathbb{P}\&\mathbb{Q}}$ where *str*, the *strength* of ! viewed as a monad on **Cts**, is the natural transformation

$$\mathbb{P} \ \& \ !\mathbb{Q} \xrightarrow{\eta_{\mathbb{P}}\&1_{!\mathbb{Q}}} !\mathbb{P} \ \& \ !\mathbb{Q} \xrightarrow{\eta_{!\mathbb{P},!\mathbb{Q}}} !\mathbb{P}\times!\mathbb{Q} \xrightarrow{m_{\mathbb{P},\mathbb{Q}}} !(\mathbb{P} \ \& \ \mathbb{Q}). \tag{13}$$

Finally, recall that the category **Lin** is symmetric monoidal closed so that the functor $(\mathbb{Q} \multimap -)$ is right adjoint to $(-\times \mathbb{Q})$ for any object $\mathbb{Q}$. Together with the natural isomorphism *m* this provides a right adjoint $(\mathbb{Q} \to -)$, defined by $(!\mathbb{Q} \multimap -)$, to the functor $(- \ \& \ \mathbb{Q})$ in **Cts** via the chain

$$\begin{aligned} \mathbf{Cts}(\mathbb{P} \ \& \ \mathbb{Q}, \mathbb{R}) &\cong \mathbf{Lin}(!(\mathbb{P} \ \& \ \mathbb{Q}), \mathbb{R}) \cong \mathbf{Lin}(!\mathbb{P}\times!\mathbb{Q}, \mathbb{R}) \\ &\cong \mathbf{Lin}(!\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) \cong \mathbf{Cts}(\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) = \mathbf{Cts}(\mathbb{P}, \mathbb{Q} \to \mathbb{R}), \end{aligned} \tag{14}$$

natural in $\mathbb{P}$ and $\mathbb{R}$. This demonstrates that **Cts** is cartesian closed, as is well known. The adjunction between **Lin** and **Cts** now satisfies the conditions put forward by Benton, Bierman, Hyland, and de Paiva for a categorical model of intuitionistic linear logic, strengthening those of Seely [4,5,49]; see also [33] for a recent survey of such models.

3. HOPLA

HOPLA is a typed process language directly suggested by the structure of the category **Cts** [42,43]. A typing judgement

$$x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q} \tag{15}$$

means that a process t yields computation paths in $\mathbb{Q}$ once processes with computation paths in $\mathbb{P}_1, \ldots, \mathbb{P}_k$ are assigned to the variables $x_1, \ldots, x_k$, respectively.

3.1. Denotational semantics

Types are given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \to \mathbb{T}_2 \mid \sum_{\alpha \in A} \mathbb{T}_\alpha \mid !\mathbb{T} \mid T \mid \mu_j \vec{T}.\vec{\mathbb{T}}. \tag{16}$$

The symbol T is drawn from a set of type variables used in defining recursive types; closed type expressions are interpreted as path orders. Using vector notation, $\mu_j \vec{T}.\vec{\mathbb{T}}$ abbreviates $\mu_j T_1, \ldots, T_k.(\mathbb{T}_1, \ldots, \mathbb{T}_k)$ and is interpreted as the j-component, for $1 \leqslant j \leqslant k$, of "the least" solution to the defining equations $T_1 = \mathbb{T}_1, \ldots, T_k = \mathbb{T}_k$, in which the expressions $\mathbb{T}_1, \ldots, \mathbb{T}_k$ may contain the T_j's. What "the least" means will be explained

below. We shall write $\mu\vec{T}.\vec{\mathbb{T}}$ as an abbreviation for the k-tuple with j-component $\mu_j\vec{T}.\vec{\mathbb{T}}$, and confuse a closed expression for a path order with the path order itself.

Simultaneous recursive equations for path orders can be solved using information systems [30,48]. Here, it will be convenient to give a concrete, inductive characterisation based on a language of *paths*

$$p,q ::= P \mapsto q \mid \beta\, p \mid P \mid abs\ p. \tag{17}$$

Above, P ranges over finite sets of paths. We use $P \mapsto q$ as notation for pairs in the function space $(!\mathbb{P})^{\mathrm{op}} \times \mathbb{Q}$. The language is complemented by formation rules using judgements $p : \mathbb{P}$, meaning that p belongs to $\mathbb{P}$, displayed below alongside rules defining the ordering on $\mathbb{P}$ using judgements $p \leqslant_{\mathbb{P}} p'$. Recall that $P \preccurlyeq_{\mathbb{P}} P'$ means $\forall p \in P.\exists p' \in P'.\ p \leqslant_{\mathbb{P}} p'$.

$$\begin{array}{ll}
\dfrac{P :!\mathbb{P} \quad q : \mathbb{Q}}{P \mapsto q : \mathbb{P} \to \mathbb{Q}} & \dfrac{P' \leqslant_{!\mathbb{P}} P \quad q \leqslant_{\mathbb{Q}} q'}{P \mapsto q \leqslant_{\mathbb{P}\to\mathbb{Q}} P' \mapsto q'} \\[2ex]
\dfrac{p : \mathbb{P}_\beta \quad \beta \in A}{\beta\, p : \sum_{\alpha\in A} \mathbb{P}_\alpha} & \dfrac{p \leqslant_{\mathbb{P}_\beta} p' \quad \beta \in A}{\beta\, p \leqslant_{\sum_{\alpha\in A}\mathbb{P}_\alpha} \beta\, p'} \\[2ex]
\dfrac{p_1 : \mathbb{P} \cdots p_n : \mathbb{P}}{\{p_1,\ldots,p_n\} : !\mathbb{P}} & \dfrac{P \preccurlyeq_{\mathbb{P}} P'}{P \leqslant_{!\mathbb{P}} P'} \\[2ex]
\dfrac{p : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{abs\ p : \mu_j\vec{T}.\vec{\mathbb{T}}} & \dfrac{p \leqslant_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} p'}{abs\ p \leqslant_{\mu_j\vec{T}.\vec{\mathbb{T}}} abs\ p'}
\end{array} \tag{18}$$

Using information systems as in [30] yields the same representation, except for the tagging with *abs* in recursive types, done to help in the proof of adequacy in Section 3.4.1. So rather than the straight equality between a recursive type and its unfolding which we are used to from [30], we get an isomorphism $abs : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \cong \mu_j\vec{T}.\vec{\mathbb{T}}$ whose inverse we call *rep*.

The raw syntax of terms is given by

$$t,u ::= x \mid rec\, x.t \mid \sum_{i\in I} t_i \mid \lambda x.t \mid t\, u \mid \beta t \mid \pi_\beta t \mid !t \mid [u > !x \Rightarrow t] \mid abs\, t \mid rep\, t. \tag{19}$$

The variable x in the "match" term $[u > !x \Rightarrow t]$ is a binding occurrence and so binds later occurrences of the variable in the body t. We shall take for granted an understanding of free and bound variables, and substitution on raw terms. The syntax will be subject to typing constraints below.

Let $\mathbb{P}_1,\ldots,\mathbb{P}_k,\mathbb{Q}$ be closed type expressions and $x_1,\ldots,x_k$ distinct variables. A syntactic judgement $x_1 : \mathbb{P}_1,\ldots,x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ stands for a map

$$[\![x_1 : \mathbb{P}_1,\ldots,x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]\!] : \mathbb{P}_1 \ \&\ \cdots\ \&\ \mathbb{P}_k \to \mathbb{Q} \tag{20}$$

in **Cts**. We will write Γ, or Λ, for an environment list $x_1 : \mathbb{P}_1,\ldots,x_k : \mathbb{P}_k$ and most often abbreviate the denotation to $\mathbb{P}_1 \ \&\ \cdots\ \&\ \mathbb{P}_k \xrightarrow{t} \mathbb{Q}$, or $\Gamma \xrightarrow{t} \mathbb{Q}$, or even $[\![t]\!]$, suppressing the type information. When the environment list is empty, the corresponding product is the empty path order $\mathbb{O}$.

The term-formation rules are displayed below alongside their interpretations as constructors on maps of **Cts**, taking the maps denoted by the premises to that denoted by the conclusion (cf. [8]). We assume that the variables in any environment list are distinct.

Structural rules: The rules handling environment lists (identity, weakening, exchange, and contraction) are given as follows:

$$\frac{}{x:\mathbb{P}\vdash x:\mathbb{P}} \qquad \frac{}{\mathbb{P}\xrightarrow{1_{\mathbb{P}}}\mathbb{P}} \tag{21}$$

$$\frac{\Gamma\vdash t:\mathbb{Q}}{\Gamma,x:\mathbb{P}\vdash t:\mathbb{Q}} \qquad \frac{\Gamma\xrightarrow{t}\mathbb{Q}}{\Gamma\ \&\ \mathbb{P}\xrightarrow{t\&\varnothing_{\mathbb{P}}}\mathbb{Q}\ \&\ \mathbb{O}\xrightarrow{r^{\mathbf{Cts}}_{\mathbb{Q}}}\mathbb{Q}} \tag{22}$$

$$\frac{\Gamma,y:\mathbb{Q},x:\mathbb{P},\Lambda\vdash t:\mathbb{R}}{\Gamma,x:\mathbb{P},y:\mathbb{Q},\Lambda\vdash t:\mathbb{R}} \qquad \frac{\Gamma\ \&\ \mathbb{Q}\ \&\ \mathbb{P}\ \&\ \Lambda\xrightarrow{t}\mathbb{R}}{\Gamma\ \&\ \mathbb{P}\ \&\ \mathbb{Q}\ \&\ \Lambda\xrightarrow{t\circ(1_\Gamma\&s^{\mathbf{Cts}}_{\mathbb{P},\mathbb{Q}}\&1_\Lambda)}\mathbb{R}} \tag{23}$$

$$\frac{\Gamma,x:\mathbb{P},y:\mathbb{P}\vdash t:\mathbb{Q}}{\Gamma,z:\mathbb{P}\vdash t[z/x,z/y]:\mathbb{Q}} \qquad \frac{\Gamma\ \&\ \mathbb{P}\ \&\ \mathbb{P}\xrightarrow{t}\mathbb{Q}}{\Gamma\ \&\ \mathbb{P}\xrightarrow{1_\Gamma\&\Delta_{\mathbb{P}}}\Gamma\ \&\ \mathbb{P}\ \&\ \mathbb{P}\xrightarrow{t}\mathbb{Q}} \tag{24}$$

In the formation rule for contraction (24), the variable z must be fresh; the map $\Delta_{\mathbb{P}}$ is the usual diagonal, given as $\langle 1_{\mathbb{P}},1_{\mathbb{P}}\rangle$.

Recursive definition: Since each $\widehat{\mathbb{P}}$ is a complete lattice, it admits least fixed-points of continuous maps. If $f:\widehat{\mathbb{P}}\to\widehat{\mathbb{P}}$ is continuous, it has a least fixed-point, $fix\, f\in\widehat{\mathbb{P}}$ obtained as $\bigcup_{n\in\omega} f^n(\varnothing)$. This allows us to interpret recursively defined processes:

$$\frac{\Gamma,x:\mathbb{P}\vdash t:\mathbb{P}}{\Gamma\vdash rec\,x.t:\mathbb{P}} \qquad \frac{\Gamma\ \&\ \mathbb{P}\xrightarrow{t}\mathbb{P}}{\Gamma\xrightarrow{fix\,F}\mathbb{P}} \tag{25}$$

Here, $fix\,F$ is the fixpoint in $\mathbf{Cts}(\Gamma,\mathbb{P})\cong\widehat{\Gamma\to\mathbb{P}}$ of the continuous operation F mapping $g:\Gamma\to\mathbb{P}$ in **Cts** to the composition

$$\Gamma\xrightarrow{\Delta_\Gamma}\Gamma\ \&\ \Gamma\xrightarrow{1_\Gamma\&g}\Gamma\ \&\ \mathbb{P}\xrightarrow{t}\mathbb{P}. \tag{26}$$

Nondeterministic sum: Each path order $\mathbb{P}$ is associated with a join operation, $\Sigma:\&_{i\in I}\mathbb{P}\to\mathbb{P}$ in **Cts** taking a tuple $\langle t_i\rangle_{i\in I}$ to the nondeterministic sum $\sum_{i\in I}t_i$ in $\widehat{\mathbb{P}}$. We will write $\varnothing$ and $t_1+\cdots+t_k$ for finite sums.

$$\frac{\Gamma\vdash t_j:\mathbb{P}\quad \text{all } j\in I}{\Gamma\vdash\sum_{i\in I}t_i:\mathbb{P}} \qquad \frac{\Gamma\xrightarrow{t_j}\mathbb{P}\quad \text{all } j\in I}{\Gamma\xrightarrow{\langle t_i\rangle_{i\in I}}\&_{i\in I}\mathbb{P}\xrightarrow{\Sigma}\mathbb{P}} \tag{27}$$

Function space: As noted at the end of Section 2.2, the category **Cts** is cartesian closed with function space $\mathbb{P}\to\mathbb{Q}$. Thus, there is a 1-1 correspondence *curry* from maps $\mathbb{P}\ \&\ \mathbb{Q}\to\mathbb{R}$ to maps $\mathbb{P}\to(\mathbb{Q}\to\mathbb{R})$ in **Cts**; its inverse is called *uncurry*. We

obtain application, $app : (\mathbb{P} \to \mathbb{Q}) \;\&\; \mathbb{P} \to \mathbb{Q}$ as $uncurry(1_{\mathbb{P}\to\mathbb{Q}})$.

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}} \qquad \frac{\Gamma \;\&\; \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\Gamma \xrightarrow{curry\, t} \mathbb{P} \to \mathbb{Q}} \tag{28}$$

$$\frac{\Gamma \vdash t : \mathbb{P} \to \mathbb{Q} \quad \Lambda \vdash u : \mathbb{P}}{\Gamma, \Lambda \vdash t\, u : \mathbb{Q}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{P} \to \mathbb{Q} \quad \Lambda \xrightarrow{u} \mathbb{P}}{\Gamma \;\&\; \Lambda \xrightarrow{t\&u} (\mathbb{P} \to \mathbb{Q}) \& \mathbb{P} \xrightarrow{app} \mathbb{Q}} \tag{29}$$

Sum type: The category **Cts** does not have coproducts, but we can build a useful sum type out of the biproduct of **Lin**. The properties (4) are obviously also satisfied in **Cts**, even though the construction is universal only in the subcategory of linear maps because composition is generally not bilinear in **Cts**. We will write $\mathbb{O}$ and $\mathbb{P}_1 + \cdots + \mathbb{P}_k$ for the empty and finite sum types. The product $\mathbb{P}_1 \;\&\; \mathbb{P}_2$ of [41] with pairing (t, u) and projection terms $fst\, t$, $snd\, t$ can be encoded as, respectively, $\mathbb{P}_1 + \mathbb{P}_2$, $1t + 2u$ and $\pi_1 t, \pi_2 t$.

$$\frac{\Gamma \vdash t : \mathbb{P}_\beta \quad \beta \in A}{\Gamma \vdash \beta t : \sum_{\alpha\in A} \mathbb{P}_\alpha} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{P}_\beta \quad \beta \in A}{\Gamma \xrightarrow{t} \mathbb{P}_\beta \xrightarrow{in_\beta} \sum_{\alpha\in A} \mathbb{P}_\alpha} \tag{30}$$

$$\frac{\Gamma \vdash t : \sum_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\Gamma \vdash \pi_\beta t : \mathbb{P}_\beta} \qquad \frac{\Gamma \xrightarrow{t} \sum_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\Gamma \xrightarrow{t} \sum_{\alpha\in A} \mathbb{P}_\alpha \xrightarrow{\pi_\beta} \mathbb{P}_\beta} \tag{31}$$

Prefixing: The adjunction between **Lin** and **Cts** provides a type constructor, $!(-)$, for which the unit $\eta_{\mathbb{P}} : \mathbb{P} \to !\mathbb{P}$ and counit $\varepsilon_{\mathbb{P}} : !\mathbb{P} \to \mathbb{P}$ play a role in interpreting term constructors and deconstructors, respectively. The behaviour of $\eta_{\mathbb{P}}$ with respect to maps of **Cts** fits that of an anonymous prefix operation. We will say that $\eta_{\mathbb{P}}$ maps u of type $\mathbb{P}$ to a "prefixed" process $!u$ of type $!\mathbb{P}$; intuitively, the process $!u$ will be able to perform an action, which we call !, before continuing as the process u.

$$\frac{\Gamma \vdash u : \mathbb{P}}{\Gamma \vdash\, !u :\, !\mathbb{P}} \qquad \frac{\Gamma \xrightarrow{u} \mathbb{P}}{\Gamma \xrightarrow{u} \mathbb{P} \xrightarrow{\eta_{\mathbb{P}}} !\mathbb{P}} \tag{32}$$

By the universal property of $\eta_{\mathbb{P}}$, if t of type $\mathbb{Q}$ has a free variable of type $\mathbb{P}$, and so is interpreted as a map $t : \mathbb{P} \to \mathbb{Q}$ in **Cts**, then the transpose $\bar{t} = \varepsilon_{\mathbb{Q}} \circ !t$ is the unique map $!\mathbb{P} \to \mathbb{Q}$ in **Lin** such that $t = \bar{t} \circ \eta_{\mathbb{P}}$. With u of type $!\mathbb{P}$, we will write $[u > !x \Rightarrow t]$ for $\bar{t}u$. Intuitively, this construction "tests" or matches u against the pattern $!x$ and passes the results of successful matches for x on to t. Indeed, first prefixing a term u of type $\mathbb{P}$ and then matching yields a successful match u for x as $\bar{t}(\eta_{\mathbb{P}} u) = tu$. By linearity of $\bar{t}$, the possibly multiple results of successful matches are nondeterministically summed together; the denotations of $[\sum_{i\in I} u_i > !x \Rightarrow t]$ and $\sum_{i\in I} [u_i > !x \Rightarrow t]$ are identical.

The above clearly generalises to the case where u is an open term, but if t has free variables other than x, we need to make use of the strength map given by (13), see Proposition 3.5:

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q} \quad \Lambda \vdash u :\, !\mathbb{P}}{\Gamma, \Lambda \vdash [u > !x \Rightarrow t] : \mathbb{Q}} \qquad \frac{\Gamma \;\&\; \mathbb{P} \xrightarrow{t} \mathbb{Q} \quad \Lambda \xrightarrow{u} !\mathbb{P}}{\Gamma \;\&\; \Lambda \xrightarrow{1_\Gamma \& u} \Gamma \;\&\; !\mathbb{P} \xrightarrow{str_{\Gamma,\mathbb{P}}} !(\Gamma \;\&\; \mathbb{P}) \xrightarrow{\bar{t}} \mathbb{P}} \tag{33}$$

Recursive type definitions: Folding and unfolding recursive types is accompanied by term constructors *abs* and *rep*:

$$\frac{\Gamma \vdash t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \vdash abs\, t : \mu_j\vec{T}.\vec{\mathbb{T}}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \xrightarrow{abs} \mu_j\vec{T}.\vec{\mathbb{T}}} \tag{34}$$

$$\frac{\Gamma \vdash t : \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \vdash rep\, t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \qquad \frac{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}} \xrightarrow{rep} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \tag{35}$$

3.2. Useful identities

We provide some technical results about the path semantics which are used in the proofs of full abstraction and soundness below. They are also useful for reasoning about encodings of process calculi, see Section 3.6.

Lemma 3.1 (Substitution). *Suppose* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ *and* $\Lambda \vdash u : \mathbb{P}$ *with* Γ *and* Λ *disjoint. Then* $\Gamma, \Lambda \vdash t[u/x] : \mathbb{Q}$ *with denotation given by the composition*

$$\Gamma \;\&\; \Lambda \xrightarrow{1_\Gamma \& u} \Gamma \;\&\; \mathbb{P} \xrightarrow{t} \mathbb{Q}. \tag{36}$$

Corollary 3.2. *Application amounts to substitution. In the situation of the substitution lemma, we have* $[\![(\lambda x.t)u]\!] = [\![t[u/x]]\!]$.

Corollary 3.3. *Recursion amounts to unfolding. Suppose* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}$. *Then* $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ *and* $[\![rec\, x.t]\!] = [\![t[rec\, x.t/x]]\!]$.

Proof. By renaming variables y of Γ to y' and y'' we get $\Gamma', x : \mathbb{P} \vdash t' : \mathbb{P}$ and $\Gamma'', x : \mathbb{P} \vdash t'' : \mathbb{P}$ with Γ' and Γ'' disjoint. Then by the substitution lemma, $\Gamma', \Gamma'' \vdash t'[rec\, x.t''/x] : \mathbb{P}$ with denotation given by

$$\Gamma' \;\&\; \Gamma'' \xrightarrow{1_{\Gamma'} \& rec\, x.t''} \Gamma' \;\&\; \mathbb{P} \xrightarrow{t'} \mathbb{P}. \tag{37}$$

By suitable use of exchange and contraction, substituting y for y' and y'', we get $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ with denotation

$$\Gamma \xrightarrow{\Delta_\Gamma} \Gamma \;\&\; \Gamma \xrightarrow{1_\Gamma \& rec\, x.t} \Gamma \;\&\; \mathbb{P} \xrightarrow{t} \mathbb{P}. \tag{38}$$

This is the same as $F(fix\, F)$ where $fix\, F$ is the denotation of $rec\, x.t$, and by property of the fixed-point, $F(fix\, F) = fix\, F$ as wanted. □

Proposition 3.4. *From the properties of the biproduct we get*

$$\begin{aligned} &[\![\pi_\beta(\beta t)]\!] = [\![t]\!], \\ &[\![\pi_\alpha(\beta t)]\!] = \varnothing \quad \text{if } \alpha \neq \beta, \\ &[\![\textstyle\sum_{\alpha\in A} \alpha(\pi_\alpha(t))]\!] = [\![t]\!], \quad \text{where } \Gamma \vdash t : \sum_{\alpha\in A} \mathbb{P}_\alpha. \end{aligned} \tag{39}$$

In addition, $[\![\beta(\sum_{i\in I} t_i)]\!] = [\![\sum_{i\in I}(\beta t_i)]\!]$ *and* $[\![\pi_\beta(\sum_{i\in I} t_i)]\!] = [\![\sum_{i\in I}(\pi_\beta t_i)]\!]$ *by linearity of injection and projection.*

Proposition 3.5. *The prefix match satisfies the properties*

$$\begin{aligned}&[\![[!u > !x \Rightarrow t]]\!] = [\![t[u/x]]\!],\\ &[\![[\sum_{i\in I} u_i > !x \Rightarrow t]]\!] = [\![\sum_{i\in I}[u_i > !x \Rightarrow t]]\!].\end{aligned} \tag{40}$$

Proof. By the properties of *str* and $\bar{t}$, and using the substitution lemma, we have

$$\begin{aligned}[!u > !x \Rightarrow t] &= \bar{t}\circ str_{\Gamma,\mathbb{P}}\circ(1_\Gamma \;\&\; (\eta_{\mathbb{P}}\circ u))\\ &= \bar{t}\circ str_{\Gamma,\mathbb{P}}\circ(1_\Gamma \;\&\; \eta_{\mathbb{P}})\circ(1_\Gamma \;\&\; u)\\ &= \bar{t}\circ \eta_{\Gamma\&\mathbb{P}}\circ(1_\Gamma \;\&\; u)\\ &= t\circ(1_\Gamma \;\&\; u)\\ &= t[u/x].\end{aligned} \tag{41}$$

Note that we are, e.g. abbreviating $[\![t]\!]$ to t. Linearity of $\bar{t}$ and $m_{\Gamma,\mathbb{P}}$ and naturality of n yields

$$\begin{aligned}\Big[\sum_{i\in I} u_i > !x \Rightarrow t\Big] &= \bar{t}\circ str_{\Gamma,\mathbb{P}}\circ\left(1_\Gamma \;\&\; \sum_{i\in I} u_i\right)\\ &= \bar{t}\circ m_{\Gamma,\mathbb{P}}\circ n_{!\Gamma,!\mathbb{P}}\circ(\eta_\Gamma \;\&\; 1_{!\mathbb{P}})\circ\left(1_\Gamma \;\&\; \sum_{i\in I} u_i\right)\\ &= \bar{t}\circ m_{\Gamma,\mathbb{P}}\circ n_{!\Gamma,!\mathbb{P}}\circ\left(\eta_\Gamma \;\&\; \sum_{i\in I} u_i\right)\\ &= \bar{t}\circ m_{\Gamma,\mathbb{P}}\circ(\eta_\Gamma \times \sum_{i\in I} u_i)\circ n_{\Gamma,\Lambda}\\ &= \sum_{i\in I}(\bar{t}\circ m_{\Gamma,\mathbb{P}}\circ(\eta_\Gamma \times u_i)\circ n_{\Gamma,\Lambda})\\ &= \sum_{i\in I}(\bar{t}\circ m_{\Gamma,\mathbb{P}}\circ n_{!\Gamma,!\mathbb{P}}\circ(\eta_\Gamma \;\&\; u_i))\\ &= \sum_{i\in I}[u_i > !x \Rightarrow t],\end{aligned} \tag{42}$$

as wanted. □

3.3. Full abstraction

We define a program to be a closed term t of type $!\mathbb{O}$, the simplest type with at least two values. A $(\Gamma,\mathbb{P})$-program context C is a term with holes into which a term t with $\Gamma\vdash t:\mathbb{P}$ may be put to form a program $\vdash C(t):!\mathbb{O}$. The denotational semantics gives rise to a type-respecting contextual preorder [38]:

Definition 3.6. Suppose $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$. We say that t_1 and t_2 are related by *contextual preorder*, written $t_1 \sqsubseteq\!\!\!\sim t_2$, iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $[\![C(t_1)]\!] \neq \emptyset \Rightarrow [\![C(t_2)]\!] \neq \emptyset$. If both $t_1 \sqsubseteq\!\!\!\sim t_2$ and $t_2 \sqsubseteq\!\!\!\sim t_1$, we say that t_1 and t_2 are *contextually equivalent*.

Contextual equivalence coincides with path equivalence, as do the associated preorders:

Theorem 3.7 (Full abstraction). *Suppose* $\Gamma \vdash t_1 : \mathbb{P}$ *and* $\Gamma \vdash t_2 : \mathbb{P}$. *Then*

$$[\![t_1]\!] \subseteq [\![t_2]\!] \;\Leftrightarrow\; t_1 \sqsubseteq\!\!\!\sim t_2. \tag{43}$$

Proof. Suppose $[\![t_1]\!] \subseteq [\![t_2]\!]$ and let C be a $(\Gamma, \mathbb{P})$-program context with $[\![C(t_1)]\!] \neq \emptyset$. As $[\![t_1]\!] \subseteq [\![t_2]\!]$ we have $[\![C(t_2)]\!] \neq \emptyset$ by compositionality and monotonicity, and so $t_1 \sqsubseteq\!\!\!\sim t_2$ as wanted.

To prove the converse we define for each path $p : \mathbb{P}$ a closed term t_p of type $\mathbb{P}$ and an $(\mathbb{O}, \mathbb{P})$-program context C_p that, respectively, "realise" and "consume" the path p, by induction on the structure of p.[3] We will also need realisers t'_P and consumers C'_P of finite sets of paths:

$$\begin{aligned}
&t_{P \mapsto q} \equiv_{\text{def}} \lambda x.[C'_P(x) > !x' \Leftrightarrow t_q], \quad C_{P \mapsto q} \equiv_{\text{def}} C_q(-t'_P),\\
&t_{\beta p} \equiv_{\text{def}} \beta t_p, \quad C_{\beta p} \equiv_{\text{def}} C_p(\pi_\beta -),\\
&t_P \equiv_{\text{def}} !t'_P, \quad C_P \equiv_{\text{def}} [- > !x \Rightarrow C'_P(x)],\\
&t_{abs\ p} \equiv_{\text{def}} abs\, t_p, \quad C_{abs\ p} \equiv_{\text{def}} C_p(rep -),\\
&t'_{\{p_1,\ldots,p_n\}} \equiv_{\text{def}} t_{p_1} + \cdots + t_{p_n},\\
&C'_{\{p_1,\ldots,p_n\}} \equiv_{\text{def}} [C_{p_1} > !x_1 \Rightarrow \cdots \Rightarrow [C_{p_n} > !x_n \Rightarrow !\emptyset] \cdots].
\end{aligned} \tag{44}$$

Note that $t'_\emptyset \equiv \emptyset$ and $C'_\emptyset \equiv !\emptyset$. Although the syntax of t'_P and C'_P depends on a choice of permutation of the elements of P, the semantics obtained for different permutations is the same. Indeed, we have (z being a fresh variable):

$$\begin{aligned}
&[\![t_p]\!] = \mathrm{y}_{\mathbb{P}}\, p \quad [\![\lambda z.C_p(z)]\!] = \mathrm{y}_{\mathbb{P} \to !\mathbb{O}}\, (\{p\} \mapsto \emptyset),\\
&[\![t'_P]\!] = i_{\mathbb{P}} P \quad [\![\lambda z.C'_P(z)]\!] = \mathrm{y}_{\mathbb{P} \to !\mathbb{O}}\, (P \mapsto \emptyset).
\end{aligned} \tag{45}$$

It then follows from the substitution lemma that for any $p : \mathbb{P}$ and $\vdash t : \mathbb{P}$,

$$p \in [\![t]\!] \;\Leftrightarrow\; [\![C_p(t)]\!] \neq \emptyset. \tag{46}$$

[3] We have recently become aware that this technique has been applied by McCusker [32] to prove full abstraction for a version of Idealized Algol.

Suppose $t_1 \sqsubseteq\!\!\!\sim t_2$ with t_1 and t_2 closed. Given any $p \in [\![t_1]\!]$ we have $[\![C_p(t_1)]\!] \neq \varnothing$ and so using $t_1 \sqsubseteq\!\!\!\sim t_2$, we get $[\![C_p(t_2)]\!] \neq \varnothing$, so that $p \in [\![t_2]\!]$. It follows that $[\![t_1]\!] \subseteq [\![t_2]\!]$.

As for open terms, suppose $\Gamma \equiv x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$. Writing $\lambda\vec{x}.t_1$ for the closed term $\lambda x_1. \cdots \lambda x_k.t_1$ and likewise for t_2, we get

$$\begin{aligned} t_1 \sqsubseteq\!\!\!\sim t_2 &\Rightarrow \lambda\vec{x}.t_1 \sqsubseteq\!\!\!\sim \lambda\vec{x}.t_2 \\ &\Rightarrow [\![\lambda\vec{x}.t_1]\!] \subseteq [\![\lambda\vec{x}.t_2]\!] \\ &\Rightarrow [\![t_1]\!] \subseteq [\![t_2]\!]. \end{aligned} \tag{47}$$

The proof is complete. □

3.4. Operational semantics

HOPLA can be given a straightforward operational semantics [43] using *actions* defined by the grammar

$$a ::= u \mapsto a \mid \beta a \mid ! \mid abs\, a. \tag{48}$$

We assign types to actions a using a judgement of the form $\mathbb{P} : a : \mathbb{P}'$. Intuitively, after performing the action a, what remains of a computation path in $\mathbb{P}$ is a computation path in $\mathbb{P}'$:

$$\frac{\vdash u : \mathbb{P} \quad \mathbb{Q} : a : \mathbb{P}'}{\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}'} \quad \frac{\mathbb{P}_\beta : a : \mathbb{P}' \quad \beta \in A}{\sum_{\alpha \in A} \mathbb{P}_\alpha : \beta a : \mathbb{P}'} \quad \frac{}{!\mathbb{P} : ! : \mathbb{P}} \quad \frac{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : a : \mathbb{P}'}{\mu_j\vec{T}.\vec{\mathbb{T}} : abs\, a : \mathbb{P}'} \tag{49}$$

Notice that in $\mathbb{P} : a : \mathbb{P}'$, the type $\mathbb{P}'$ is unique given $\mathbb{P}$ and a. The operational rules of Fig. 1 define a relation $\mathbb{P} : t \xrightarrow{a} t'$ where $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$. By rule induction on the transition rules, we have

Proposition 3.8. *If* $\mathbb{P} : t \xrightarrow{a} t'$ *with* $\mathbb{P} : a : \mathbb{P}'$, *then* $\vdash t' : \mathbb{P}'$.

Accordingly, we will write $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ when $\mathbb{P} : t \xrightarrow{a} t'$ and $\mathbb{P} : a : \mathbb{P}'$.

3.4.1. Soundness and adequacy

For $\mathbb{P} : a : \mathbb{P}'$, we define a linear map $a^* : \mathbb{P} \to !\mathbb{P}'$ which intuitively maps a process t of type $\mathbb{P}$ to a representation of its possible successors after performing the action a. In order to distinguish between, say, the successor $\varnothing$ and no successors, a^* embeds into the type $!\mathbb{P}'$ rather than using $\mathbb{P}'$ itself. For instance, the successors after action $!$ of the processes $!\varnothing$ and $\varnothing$ are, respectively,

$$!^*[\![!\varnothing]\!] = 1_{!\mathbb{P}}(\eta_{\mathbb{P}}\varnothing) = \eta_{\mathbb{P}}\varnothing \quad \text{and} \quad !^*[\![\varnothing]\!] = 1_{!\mathbb{P}}\varnothing = \varnothing. \tag{50}$$

$$\frac{\mathbb{P} : t[rec\,x.t/x] \xrightarrow{a} t'}{\mathbb{P} : rec\,x.t \xrightarrow{a} t'} \qquad \frac{\mathbb{P} : t_j \xrightarrow{a} t'}{\mathbb{P} : \sum_{i\in I} t_i \xrightarrow{a} t'} j \in I$$

$$\frac{\mathbb{Q} : t[u/x] \xrightarrow{a} t'}{\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u\mapsto a} t'} \qquad \frac{\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u\mapsto a} t'}{\mathbb{Q} : t\,u \xrightarrow{a} t'}$$

$$\frac{\mathbb{P}_\beta : t \xrightarrow{a} t'}{\sum_{\alpha\in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta a} t'} \qquad \frac{\sum_{\alpha\in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t'}{\mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t'}$$

$$\frac{}{!\mathbb{P} : !t \xrightarrow{!} t} \qquad \frac{!\mathbb{P} : u \xrightarrow{!} u' \quad \mathbb{Q} : t[u'/x] \xrightarrow{a} t'}{\mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{a} t'}$$

$$\frac{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{a} t'}{\mu_j\vec{T}.\vec{\mathbb{T}} : abs\,t \xrightarrow{abs\,a} t'} \qquad \frac{\mu_j\vec{T}.\vec{\mathbb{T}} : t \xrightarrow{abs\,a} t'}{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\,t \xrightarrow{a} t'}$$

Fig. 1. Operational rules.

It will be convenient to treat a^* as a syntactic operation and so we define a term a^*t such that $[\![a^*t]\!] = a^*[\![t]\!]$:

$$\begin{array}{rlrl} (u \mapsto a)^* &= a^* \circ app \circ (- \;\&\; [\![u]\!]), & (u \mapsto a)^*t &\equiv a^*(t\,u), \\ (\beta a)^* &= a^* \circ \pi_\beta, & (\beta a)^*t &\equiv a^*(\pi_\beta t), \\ !^* &= 1_{!\mathbb{P}}, & !^*t &\equiv t, \\ (abs\,a)^* &= a^* \circ rep, & (abs\,a)^*t &\equiv a^*(rep\,t). \end{array} \tag{51}$$

The syntactic operation a^* can be viewed as providing a context which reduces a-transitions to !-transitions.

Lemma 3.9. $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \Leftrightarrow !\mathbb{P}' : a^*t \xrightarrow{!} t' : \mathbb{P}'$.

Proof. By structural induction on actions. For the prefix action ! the result is immediate. We present the case for $u \mapsto a$, the two remaining cases being similar. As there is only one operational rule deriving transitions for applications $t\,u$ we have

$$\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u\mapsto a} t' : \mathbb{P}' \Leftrightarrow \mathbb{Q} : t\,u \xrightarrow{a} t' : \mathbb{P}'. \tag{52}$$

By the induction hypothesis, the right-hand side is equivalent to $!\mathbb{P}' : a^*(t\,u) \xrightarrow{!} t' : \mathbb{P}'$, and by definition of $(u \mapsto a)^*t$ we are done. □

Writing $\mathbb{P} : t \xrightarrow{a}$ when there exists t' such that $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$, the following are equivalent:

$$\text{(i) } \mathbb{P} : t \xrightarrow{a} \qquad \text{(ii) } !\mathbb{P}' : a^*t \xrightarrow{!} \qquad \text{(iii) } !\mathbb{O} : C_\varnothing(a^*t) \xrightarrow{!}. \tag{53}$$

Here, $C_\varnothing$ is the $(\mathbb{O}, !\mathbb{P}')$-program context $[\Rightarrow !x \to !\varnothing]$ from the proof of full abstraction.

Thus, observations of general transitions and !-transitions are reducible to observations of !-transitions at type $!\mathbb{O}$. We will exploit this below to give an operational formulation of full abstraction.

Proposition 3.10 (Soundness). *If* $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$, *then* $[\![!t']\!] \subseteq a^*[\![t]\!]$.

Proof. By rule-induction on the transition rules, see Appendix A. □

We obtain an adequacy result using logical relations $X \trianglelefteq_{\mathbb{P}} t$ between subsets $X \subseteq \mathbb{P}$ and closed terms of type $\mathbb{P}$. Intuitively, $X \trianglelefteq_{\mathbb{P}} t$ means that all paths in X can be "operationally realised" by t. Because of recursive types, these relations cannot be defined by structural induction on the type $\mathbb{P}$ and we therefore employ a trick essentially due to Martin–Löf (see [53]). We define auxiliary relations $p\,\varepsilon_{\mathbb{P}}\,t$ between paths $p : \mathbb{P}$ and closed terms t of type $\mathbb{P}$, by induction on the structure of p:

$$X \trianglelefteq_{\mathbb{P}} t \quad \Leftrightarrow_{\text{def}} \quad \forall p \in X.\ p\,\varepsilon_{\mathbb{P}}\,t, \tag{54}$$

$$P \mapsto q\,\varepsilon_{\mathbb{P}\to\mathbb{Q}}\,t \quad \Leftrightarrow_{\text{def}} \quad \forall u.(P \trianglelefteq_{\mathbb{P}} u \Rightarrow q\,\varepsilon_{\mathbb{Q}}\,t\,u), \tag{55}$$

$$\beta\,p\,\varepsilon_{\sum_{\alpha\in A}\mathbb{P}_\alpha}\,t \quad \Leftrightarrow_{\text{def}} \quad p\,\varepsilon_{\mathbb{P}_\beta}\,\pi_\beta t, \tag{56}$$

$$P\,\varepsilon_{!\mathbb{P}}\,t \quad \Leftrightarrow_{\text{def}} \quad \exists t'.!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P} \text{ and } P \trianglelefteq_{\mathbb{P}} t', \tag{57}$$

$$abs\ p\,\varepsilon_{\mu_j\vec{T}.\vec{\mathbb{T}}}\,t \quad \Leftrightarrow_{\text{def}} \quad p\,\varepsilon_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}\,rep\,t. \tag{58}$$

Lemma 3.11 (Main lemma). *Suppose* $\vdash t : \mathbb{P}$. *Then* $[\![t]\!] \trianglelefteq_{\mathbb{P}} t$.

Proof. By structural induction on terms, see Appendix B. □

Proposition 3.12 (Adequacy). *For* $\vdash t : !\mathbb{P}$ *we have* $[\![t]\!] \neq \varnothing \Leftrightarrow !\mathbb{P} : t \xrightarrow{!}$.

Proof. The "$\Leftarrow$" direction follows from soundness. Assume $[\![t]\!] \neq \varnothing$. Then because $[\![t]\!]$ is a downwards-closed subset of $!\mathbb{P}$ which has least element $\varnothing$, we must have $\varnothing \in [\![t]\!]$. Thus $\varnothing\,\varepsilon_{!\mathbb{P}}\,t$ by Lemma 3.11, which implies the existence of a term t' such that $!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P}$ as wanted. □

By (53), adequacy is equivalent to $a^*[\![t]\!] \neq \varnothing \Leftrightarrow \mathbb{P} : t \xrightarrow{a}$ for general terms $\vdash t : \mathbb{P}$.

3.4.2. Full abstraction w.r.t. operational semantics

Adequacy allows an operational formulation of contextual equivalence. For programs $\vdash t : !\mathbb{O}$ we have $!\mathbb{O} : t \xrightarrow{!}$ iff $[\![t]\!] \neq \varnothing$ by adequacy. Hence, two terms t_1 and t_2 with $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$ are related by contextual preorder iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $!\mathbb{O} : C(t_1) \xrightarrow{!} \Rightarrow !\mathbb{O} : C(t_2) \xrightarrow{!}$.

Full abstraction is often formulated in terms of this operational preorder. With t_1 and t_2 as above, the inclusion $[\![t_1]\!] \subseteq [\![t_2]\!]$ holds iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $!\mathbb{O} : C(t_1) \xrightarrow{!} \Rightarrow !\mathbb{O} : C(t_2) \xrightarrow{!}$.

3.5. Simulation

The operational semantics supports a standard bisimulation [35,44].

Definition 3.13. A type-respecting relation R on closed terms is a *bisimulation* if the following holds. If $t_1 R t_2$ with t_1, t_2 of the same type $\mathbb{P}$, then
1. if $\mathbb{P} : t_1 \xrightarrow{a} t_1' : \mathbb{P}'$, then $\mathbb{P} : t_2 \xrightarrow{a} t_2' : \mathbb{P}'$ for some t_2' such that $t_1' \, R \, t_2'$;
2. if $\mathbb{P} : t_2 \xrightarrow{a} t_2' : \mathbb{P}'$, then $\mathbb{P} : t_1 \xrightarrow{a} t_1' : \mathbb{P}'$ for some t_1' such that $t_1' \, R \, t_2'$.
Bisimilarity, written $\sim$, is the largest bisimulation.

Bisimilarity is a congruence for HOPLA and coincides with notions of applicative bisimilarity [1] and higher-order bisimilarity [51]—see [42].

The path semantics does not capture enough of the branching behaviour of processes to characterise bisimilarity (for that, the presheaf semantics is needed, see Section 6.1). As an example, the processes $!\varnothing + !!\varnothing$ and $!!\varnothing$ have the same denotation, but are clearly not bisimilar. However, using Hennessy–Milner logic we can link path equivalence to *simulation*, obtained as in Definition 3.13, but leaving out condition 2. In detail, we consider the fragment of Hennessy–Milner logic given by possibility and finite conjunctions; it is characteristic for simulation equivalence in the case of image-finite processes [21]. With a ranging over actions, formulae are given by

$$\phi ::= \langle a \rangle \phi \mid \bigwedge_{i \leqslant n} \phi_i. \tag{59}$$

The empty conjunction is written $\top$ and we sometimes write $\phi_1 \wedge \cdots \wedge \phi_n$ for the conjunction $\bigwedge_{i \leqslant n} \phi_i$. We type formulae using judgements $\phi : \mathbb{P}$, the idea being that only processes of type $\mathbb{P}$ should be described by formulae of type $\mathbb{P}$:

$$\frac{\mathbb{P} : a : \mathbb{P}' \quad \phi : \mathbb{P}'}{\langle a \rangle \phi : \mathbb{P}} \qquad \frac{\phi_i : \mathbb{P} \quad \text{all } i \leqslant n}{\bigwedge_{i \leqslant n} \phi_i : \mathbb{P}} \tag{60}$$

A typed notion of satisfaction, written $t \vDash \phi : \mathbb{P}$, is defined by

$$\begin{aligned} &t \vDash \langle a \rangle \phi : \mathbb{P} \Leftrightarrow_{\text{def}} \exists t'.\ \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \quad \text{and} \quad t' \vDash \phi : \mathbb{P}', \\ &t \Leftrightarrow \bigwedge_{i \leqslant n} \phi_i : \mathbb{P} \Leftrightarrow_{\text{def}} t \vDash \phi_i : \mathbb{P} \text{ for each } i \leqslant n. \end{aligned} \tag{61}$$

Note that $\top : \mathbb{P}$ and $t \vDash \top : \mathbb{P}$ for all terms $\vdash t : \mathbb{P}$.

Definition 3.14. Suppose $\vdash t_1 : \mathbb{P}$ and $\vdash t_2 : \mathbb{P}$. We say that t_1 and t_2 are related by the *logical preorder*, written $t_1 \lesssim_{\mathrm{L}} t_2$, iff for all formulae $\phi : \mathbb{P}$ we have $t_1 \vDash \phi : \mathbb{P} \Rightarrow t_2 \vDash \phi : \mathbb{P}$. If both $t_1 \lesssim_{\mathrm{L}} t_2$ and $t_2 \lesssim_{\mathrm{L}} t_1$, we say that t_1 and t_2 are *logically equivalent*.

Using adequacy and adapting the proof of full abstraction, we can show that logical equivalence coincides with contextual equivalence.

Theorem 3.15. *For closed terms t_1 and t_2 of the same type $\mathbb{P}$,*

$$t_1 \sqsubseteq\!\!\!\sim t_2 \;\Leftrightarrow\; t_1 \sqsubseteq\!\!\!\sim_{\mathrm{L}} t_2. \tag{62}$$

Proof. To each formula $\phi : \mathbb{P}$, we can construct an $(\mathbb{O}, \mathbb{P})$-program context C_ϕ with the property that

$$!\mathbb{O} : C_\phi(t) \xrightarrow{\;!\;} \Leftrightarrow t \vDash \phi : \mathbb{P}. \tag{63}$$

Define

$$\begin{array}{l} C_{\langle u \mapsto a\rangle\phi} \equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(-u), C_{\langle !\rangle\phi} \equiv_{\mathrm{def}} [- > !x \Rightarrow C_\phi(x)], \\ C_{\langle \beta a\rangle\phi} \equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(\pi_\beta -), C_{\langle abs\, a\rangle\phi} \equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(rep\, -), \\ C_{\bigwedge_{i\leqslant n}\phi_i} \equiv_{\mathrm{def}} [C_{\phi_1} > !x_1 \Rightarrow \cdots \Rightarrow [C_{\phi_n} > !x_n \Rightarrow !\varnothing]\cdots]. \end{array} \tag{64}$$

It follows by (63) that $t_1 \sqsubseteq\!\!\!\sim_{\mathrm{L}} t_2$ iff for all formulae $\phi : \mathbb{P}$ we have that $!\mathbb{O} : C_\phi(t_1) \xrightarrow{!}$ implies $!\mathbb{O} : C_\phi(t_2) \xrightarrow{!}$. The direction "$\Rightarrow$" then follows by adequacy.

For the converse, we observe that the program contexts C_p used in the full-abstraction proof are all subsumed by the contexts C_ϕ. In detail, using the terms t'_P realising finite sets of paths, we can define actions $\mathbb{P} : a_p : \mathbb{P}'$ and formulae $\phi_p : \mathbb{P}$ by induction on paths $p : \mathbb{P}$ such that $C_p \equiv C_{\langle a_p\rangle\phi_p}$:

$$\begin{array}{rlrl} a_{P\mapsto q} &\equiv_{\mathrm{def}} t'_P \mapsto a_q & \phi_{P\mapsto q} &\equiv_{\mathrm{def}} \phi_q \\ a_{\beta p} &\equiv_{\mathrm{def}} \beta a_p & \phi_{\beta p} &\equiv_{\mathrm{def}} \phi_p \\ a_P &\equiv_{\mathrm{def}} ! & \phi_P &\equiv_{\mathrm{def}} \bigwedge_{p\in P} \langle a_p\rangle\phi_p \\ a_{abs\, p} &\equiv_{\mathrm{def}} abs\, a_p & \phi_{abs\, p} &\equiv_{\mathrm{def}} \phi_p \end{array} \tag{65}$$

With $p : \mathbb{P}$ and $\vdash t : \mathbb{P}$ we obtain $p \in [\![t]\!] \Leftrightarrow [\![C_{\langle a_p\rangle\phi_p}(t)]\!] \neq \varnothing$ as in the proof of full abstraction, and so by adequacy and (63), we have $p \in [\![t]\!] \Leftrightarrow t \vDash \langle a_p\rangle\phi_p : \mathbb{P}$. It follows that $t_1 \sqsubseteq\!\!\!\sim_{\mathrm{L}} t_2$ implies $[\![t_1]\!] \subseteq [\![t_2]\!]$, and so $t_1 \sqsubseteq\!\!\!\sim t_2$. □

We note that the proof above establishes a link between paths and actions:

$$p \in [\![t]\!] \Leftrightarrow \mathbb{P} : t \xrightarrow{a_p} t' : \mathbb{P}' \quad \text{and} \quad t' \vDash \phi_p : \mathbb{P}'. \tag{66}$$

3.6. Expressive power

HOPLA does not have many features typical of process calculi built-in, beyond that of a nondeterministic sum and a prefix operation. It is therefore notable that we can express many kinds of concurrent processes in the language. We start by encoding the

"prefix-sum" construct of [42], useful for subsequent examples. We will dispense with the abstract syntax *abs* and *rep* for brevity.

3.6.1. Prefixed sum

In the original presentation of HOPLA, prefixing and the sum type where part of a single construct, the prefixed sum [42]. Consider a family of types $(\mathbb{P}_\alpha)_{\alpha\in A}$. Their *prefixed sum* is the type $\sum_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ which stands for $\sum_{\alpha\in A}!\mathbb{P}_\alpha$. This type describes computation paths in which first an action $\beta\in A$ is performed before resuming as a computation path in $\mathbb{P}_\beta$. We will write $\alpha_1.\mathbb{P}_{\alpha_1}+\cdots+\alpha_k.\mathbb{P}_{\alpha_k}$ for a typical finite prefixed sum. The prefixed sum is associated with prefix operations taking a process t of type $\mathbb{P}_\beta$ to $\beta.t\equiv_{\mathrm{def}}\beta(!t)$ of type $\sum_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ as well as a prefix match $[u>\beta.x\Rightarrow t]\equiv_{\mathrm{def}}[\pi_\beta u>!x\Rightarrow t]$, where u has prefix-sum type, x has type $\mathbb{P}_\beta$ and t generally involves the variable x.

Proposition 3.16. *Using Propositions* 3.4 *and* 3.5, *we get*

$$[\![[\beta.u>\beta.x\Rightarrow t]]\!]=[\![t[u/x]]\!], \tag{67}$$

$$[\![[\alpha.u>\beta.x\Rightarrow t]]\!]=\varnothing \quad \textit{if } \alpha\neq\beta, \tag{68}$$

$$[\![[\sum_{i\in I}u_i>\beta.x\Rightarrow t]]\!]=[\![\sum_{i\in I}[u_i>\beta.x\Rightarrow t]]\!]. \tag{69}$$

Note that the prefixed sum is obtained using the biproduct, so coproduct, of **Lin**. This implies that prefixed sum is a "weak coproduct" in **Cts**. Because of the universal property of the coproduct in **Lin** and using the adjunction between **Lin** and **Cts**, there is a chain of isomorphisms

$$\mathbf{Lin}\left(\sum_{\alpha\in A}!\mathbb{P}_\alpha,\mathbb{Q}\right)\cong\prod_{\alpha\in A}\mathbf{Lin}(\mathbb{P}_\alpha,\mathbb{Q})\cong\prod_{\alpha\in A}\mathbf{Cts}(\mathbb{P}_\alpha,\mathbb{Q}), \tag{70}$$

natural in $\mathbb{Q}$. Hence, *linear* maps $f:\sum_{\alpha\in A}!\mathbb{P}_\alpha\to\mathbb{Q}$ from the prefixed sum in **Cts** are in bijective correspondence with tuples $\langle f_\alpha\rangle_{\alpha\in A}$ of maps from the components of the sum to $\mathbb{Q}$ in **Cts**. Thus, the prefixed sum is a coproduct in **Cts** but for the fact that the required mediating morphism is unique only within the subcategory of linear maps.

3.6.2. CCS

As in CCS [35], let N be a set of names and $\bar{N}$ the set of complemented names $\{\bar{n}\mid n\in N\}$. Let l range over labels $L=_{\mathrm{def}}N\cup\bar{N}$, with complementation extended to L by taking $\bar{\bar{n}}=_{\mathrm{def}}n$, and let τ be a distinct label. The type of CCS processes can then be specified as the solution to the equation

$$\mathbb{P}=\tau.\mathbb{P}+\sum_{n\in N}n.\mathbb{P}+\sum_{n\in N}\bar{n}.\mathbb{P}. \tag{71}$$

Below, we let α range over $L \cup \{\tau\}$. The terms of CCS are translated into HOPLA by the function $\mathscr{H}[\![-]\!]$,

$$\begin{array}{rlll} \mathscr{H}[\![x]\!] & \equiv_{\text{def}} & x, & \\ \mathscr{H}[\![rec\, x.t]\!] & \equiv_{\text{def}} & rec\, x.\mathscr{H}[\![t]\!], & \mathscr{H}[\![t \mid u]\!] \equiv_{\text{def}} Par\ \mathscr{H}[\![t]\!]\ \mathscr{H}[\![u]\!], \\ \mathscr{H}[\![\sum_{i\in I} t_i]\!] & \equiv_{\text{def}} & \sum_{i\in I} \mathscr{H}[\![t_i]\!], & \mathscr{H}[\![t\backslash S]\!] \equiv_{\text{def}} Res_S\ \mathscr{H}[\![t]\!], \\ \mathscr{H}[\![\alpha.t]\!] & \equiv_{\text{def}} & \alpha.\mathscr{H}[\![t]\!], & \mathscr{H}[\![t[f]]\!] \equiv_{\text{def}} Rel_f\ \mathscr{H}[\![t]\!]. \end{array} \tag{72}$$

Here, $Par : \mathbb{P} \to (\mathbb{P} \to \mathbb{P})$ (curried for convenience), $Res_S : \mathbb{P} \to \mathbb{P}$, and $Rel_f : \mathbb{P} \to \mathbb{P}$ are abbreviations for the following recursively defined processes:

$$\begin{aligned} Par \equiv_{\text{def}} rec\ p.\lambda x.\lambda y.\ & \sum_{\alpha} [x > \alpha.x' \Rightarrow \alpha.(p\ x'\ y)] \\ & + \sum_{\alpha} [y > \alpha.y' \Rightarrow \alpha.(p\ x\ y')] \\ & + \sum_{l} [x > l.x' \Rightarrow [y > \bar{l}.y' \Rightarrow \tau.(p\ x'\ y')]], \end{aligned} \tag{73}$$

$$Res_S \equiv_{\text{def}} rec\, r.\lambda x.\ \sum_{\alpha \notin (S \cup \bar{S})} [x > \alpha.x' \Rightarrow \alpha.(r\ x')], \tag{74}$$

$$Rel_f \equiv_{\text{def}} rec\, r.\lambda x.\ \sum_{\alpha} [x > \alpha.x' \Rightarrow f(\alpha).(r\ x')]. \tag{75}$$

The operational semantics for CCS induced by the translation agrees with that given by Milner.

Proposition 3.17. *If $t \xrightarrow{\alpha} t'$ is derivable in CCS, then $\mathbb{P} : \mathscr{H}[\![t]\!] \xrightarrow{\alpha!} \mathscr{H}[\![t']\!] : \mathbb{P}$. Conversely, if $\mathbb{P} : \mathscr{H}[\![t]\!] \xrightarrow{a} u : \mathbb{P}$, then $a \equiv \alpha!$ and $u \equiv \mathscr{H}[\![t']\!]$ for some α, t' such that $t \xrightarrow{\alpha} t'$ according to CCS.*

It follows that the translations of two CCS terms are bisimilar in HOPLA iff they are strongly bisimilar in CCS.

We can recover Milner's expansion law [34] directly from the properties of the prefixed sum. Write $t \mid u$ for the application $Par\, t\, u$, where t and u are terms of type $\mathbb{P}$. Suppose

$$[\![t]\!] = [\![\sum_{\alpha} \sum_{i\in I(\alpha)} \alpha.t_i]\!] \quad \text{and} \quad [\![u]\!] = [\![\sum_{\alpha} \sum_{j\in J(\alpha)} \alpha.u_j]\!]. \tag{76}$$

Using Corollaries 3.3 and 3.2, then Proposition 3.16, $[\![t|u]\!]$ equals the denotation of the expansion

$$\sum_{\alpha} \sum_{i\in I(\alpha)} \alpha.(t_i|u) + \sum_{\alpha} \sum_{j\in J(\alpha)} \alpha.(t|u_j) + \sum_{l} \sum_{i\in I(l), j\in J(\bar{l})} \tau.(t_i|u_j). \tag{77}$$

3.6.3. Higher-Order CCS

The language considered by Hennessy [22] is like CCS but where processes are passed at channels C; the language can be seen as an extension of Thomsen's CHOCS

[51]. For a translation into HOPLA, we follow Hennessy in defining types that satisfy the equations [4]

$$\mathbb{P} = \tau.\mathbb{P} + \sum_{c\in C} \bar{c}.\mathbb{C} + \sum_{c\in C} c.\mathbb{F}, \quad \mathbb{C} = \mathbb{P}\&\mathbb{P}, \quad \mathbb{F} = \mathbb{P} \to \mathbb{P}. \tag{78}$$

We are chiefly interested in the parallel composition of processes, $Par_{\mathbb{P},\mathbb{P}}$ of type $\mathbb{P}\&\mathbb{P} \to \mathbb{P}$. But parallel composition is really a family of mutually dependent operations also including components such as $Par_{\mathbb{F},\mathbb{C}}$ of type $\mathbb{F}\&\mathbb{C} \to \mathbb{P}$ to say how abstractions compose in parallel with concretions, etc. All these components can be tupled together in a product and parallel composition defined as a simultaneous recursive definition. Writing $(-|-)$ for all the components of the solution, the denotation of a parallel composition $t|u$ of processes equals the denotation of the expansion

$$\begin{aligned}
&\sum_{\alpha} [t > \alpha.x \Rightarrow \alpha.(x|u)] \\
&\quad + \sum_{\alpha} [u > \alpha.y \Rightarrow \alpha.(t|y)] \\
&\quad + \sum_{c} [t > c.f \Rightarrow [u > \bar{c}.p \Rightarrow \tau.((f\ \ \pi_1\ \ p)|\pi_2 p)]] \\
&\quad + \sum_{c} [t > \bar{c}.p \Rightarrow [u > c.f \Rightarrow \tau.(\pi_2 p|(f\ \ \pi_1\ \ p))]].
\end{aligned} \tag{79}$$

In the summations, $c \in C$ and α ranges over labels $c, \bar{c}, \tau$.

The bisimulation induced on higher-order CCS terms is perhaps the one to be expected; a corresponding bisimulation relation is defined like an applicative bisimulation but restricted to the types of processes $\mathbb{P}$, concretions $\mathbb{C}$, and abstractions $\mathbb{F}$.

In a similar way, we can encode Cardelli and Gordon's ambient calculus with public names [10,11], see [42]. HOPLA can thus express certain forms of mobility of processes by virtue of allowing process passing. Another kind of mobility, mobility of communication links, arises from name-generation as in the π-calculus [36]. Inspired by HOPLA, Francesco Zappa Nardelli and GW have defined a higher-order process language with name-generation, allowing encodings of full ambient calculus and π-calculus. Bisimulation properties and semantic underpinnings are being developed [54].

4. Linearity

The move from **Lin** to **Cts** has allowed us to interpret prefixing. In fact, we can do much the same more cheaply.

The category **Cts** is obtained from **Lin** using an exponential which allows arbitrary copying in linear logic. An element $P \in !\mathbb{P}$ consists of several, possibly no, computation paths of $\mathbb{P}$. An element of the path order $!\mathbb{P}$ can therefore be understood intuitively as describing a compound computation path associated with running several copies of a process of type $\mathbb{P}$. Maps $\mathbb{P} \to \mathbb{Q}$ of **Cts**, corresponding to maps $!\mathbb{P} \to \mathbb{Q}$ of **Lin**, allow

[4] See Section 3.1 (Sum type) for how to encode the binary product $\mathbb{P}$ & $\mathbb{P}$.

their input to be copied, as witnessed by the fact that the type system of HOPLA allows contraction.

However, copying is generally restricted in a distributed computation. A communication received is most often the result of a single run of the process communicated with. Of course, process code can be sent and copied. But, generally, the receiver has no possibility of rewinding or copying the state of an ongoing computation. On the other hand, ignoring another process is often easy. For this reason, many operations of distributed computation have the following property [41]:

Affine linearity: a computation path of the process arising from the application of an operation to an input process has resulted from at most one computation path of the input process.

Note, in particular, that prefix operations are affine in this sense: if we wish to observe just the initial action of a process $!t$, no computation path of t is needed, though observing any longer path will involve a (single) computation path of t.

Recall the diagram (2) which says that linear maps $\mathbb{P} \to \mathbb{Q}$ are determined by their values on single paths, elements of $\mathbb{P}$. Via the adjunction between **Lin** and **Cts**, continuous maps $\mathbb{P} \to \mathbb{Q}$ are determined by their values on compound paths in $!\mathbb{P}$ (diagram (7)). To summarise:

- linear operations use *a single* path of the input;
- affine operations use *at most one* path of the input;
- continuous operations use *any number of* paths of the input.

Affine maps are defined by their values on singleton copies of paths together with the empty path. Accordingly, affine maps derive from the lifting operation $(-)_\perp$ adding a new element $\perp$, to be thought of as the empty computation path, below a copy of a path order $\mathbb{P}$ to produce a path order $\mathbb{P}_\perp$. Abstractly, $\mathbb{P}_\perp$ is the empty-join completion of $\mathbb{P}$; concretely, we can take $\mathbb{P}_\perp$ to contain the empty set, written $\perp$, together with singletons $\{p\}$ for $p \in \mathbb{P}$, ordered by $\preccurlyeq_{\mathbb{P}}$. There is an obvious inclusion of the empty-join completion of $\mathbb{P}$ into $\widehat{\mathbb{P}}$, in the form of a map $j_{\mathbb{P}} : \mathbb{P}_\perp \to \widehat{\mathbb{P}}$ sending $\perp$ to $\varnothing$ and $\{p\}$ to $\mathrm{y}_{\mathbb{P}}\, p$. We will use P to range over $\mathbb{P}_\perp$ in what follows. The map $j_{\mathbb{P}}$ assumes the role of $i_{\mathbb{P}}$; for any $X \in \widehat{\mathbb{P}}$ and $P \in \mathbb{P}_\perp$ we have $j_{\mathbb{P}} P \subseteq X$ iff $P \preccurlyeq_{\mathbb{P}} X$, and from (1) we get

$$X = \bigcup_{p \in X} \mathrm{y}_{\mathbb{P}}\, p = \varnothing \cup \bigcup_{p \in X} \mathrm{y}_{\mathbb{P}}\, p = \bigcup_{P \preccurlyeq_{\mathbb{P}} X} j_{\mathbb{P}} P. \tag{80}$$

This join is manifestly nonempty and in fact, $\widehat{\mathbb{P}}$ is the free closure of $\mathbb{P}_\perp$ under nonempty joins. This means that given any monotone map $f : \mathbb{P}_\perp \to C$ for some nonempty-join complete poset C, there is a unique nonempty-join preserving (i.e. *affine*) map $f^{\S} : \widehat{\mathbb{P}} \to C$ such that the diagram below commutes:

$$\begin{array}{ccc} \mathbb{P}_\perp & \xrightarrow{j_{\mathbb{P}}} & \widehat{\mathbb{P}} \\ & \searrow^{f} & \downarrow f^{\S} \\ & & C \end{array} \qquad f^{\S} X = \bigcup_{P \preccurlyeq_{\mathbb{P}} X} f P. \tag{81}$$

Uniqueness of $f^{\S}$, called the *extension of f along $j_{\mathbb{P}}$*, follows from (80). As before, we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (2) and (81), there is a bijective correspondence between linear maps $\mathbb{P}_\perp \to \mathbb{Q}$ and affine maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

We define the category **Aff** to have path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and affine maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. Again, the structure of **Aff** is induced by that of **Lin** via an adjunction between the two categories with the inclusion **Lin** $\hookrightarrow$ **Aff** (linear maps are affine) as right adjoint:

$$\mathbf{Lin}(\mathbb{P}_\perp, \mathbb{Q}) \cong \mathbf{Aff}(\mathbb{P}, \mathbb{Q}). \tag{82}$$

The unit $\eta_{\mathbb{P}} : \mathbb{P} \to \mathbb{P}_\perp$ in **Aff**, the counit $\varepsilon_{\mathbb{P}} : \mathbb{P}_\perp \to \mathbb{P}$ in **Lin**, and the left adjoint $(-)_\perp : \mathbf{Aff} \to \mathbf{Lin}$ are obtained precisely as in Section 2.2.

Aff inherits products $\sum_{\alpha \in A} \mathbb{P}_\alpha$ with weak coproduct properties from **Lin** in the same way as **Cts** does. However, unlike **Cts**, the category **Aff** is not cartesian closed because $\mathbb{P}_\perp \times \mathbb{Q}_\perp$ and $(\mathbb{P}\&\mathbb{Q})_\perp$ are not isomorphic in **Lin**. On the other hand, we can easily define a tensor operation $\otimes$ on **Aff** such that the path orders $\mathbb{P}_\perp \times \mathbb{Q}_\perp$ and $(\mathbb{P} \otimes \mathbb{Q})_\perp$ become isomorphic: simply take $\mathbb{P} \otimes \mathbb{Q}$ to be $(\mathbb{P}_\perp \times \mathbb{Q}_\perp) \backslash \{(\perp, \perp)\}$. Paths of $\mathbb{P} \otimes \mathbb{Q}$ then consist of a (possibly empty) path of $\mathbb{P}$ and a (possibly empty) path of $\mathbb{Q}$, and so a path set $X \in \widehat{\mathbb{P} \otimes \mathbb{Q}}$ can be thought of as a process performing two parallel computation paths, one of type $\mathbb{P}$ and one of type $\mathbb{Q}$. On arrows $f : \mathbb{P} \to \mathbb{P}'$ and $g : \mathbb{Q} \to \mathbb{Q}'$ in **Aff**, we define $f \otimes g : \mathbb{P} \otimes \mathbb{Q} \to \mathbb{P}' \otimes \mathbb{Q}'$ as the extension $h^{\S}$ of the map $h : \mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P} \otimes \mathbb{Q})_\perp \to \widehat{\mathbb{P}' \otimes \mathbb{Q}'}$ defined by

$$(P', Q') \in h(P, Q) \iff P' \in (\eta_{\mathbb{P}'} \circ f \circ j_{\mathbb{P}})P \quad \text{and} \quad Q' \in (\eta_{\mathbb{Q}'} \circ g \circ j_{\mathbb{Q}})Q. \tag{83}$$

The unit of tensor is the empty path order $\mathbb{O}$. Elements $X \in \widehat{\mathbb{P}}$ correspond to maps $\bar{X} : \mathbb{O} \to \mathbb{P}$ in **Aff** and with $Y \in \widehat{\mathbb{Q}}$, we will write $X \otimes Y$ for the element of $\widehat{\mathbb{P} \otimes \mathbb{Q}}$ pointed to by the map $\bar{X} \otimes \bar{Y}$. The tensor makes **Aff** a symmetric monoidal category, and again, adjunction (82) is symmetric monoidal. The obvious isomorphisms of path orders,

$$\mathbb{1} \cong \mathbb{O}_\perp \quad \text{and} \quad \mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P} \otimes \mathbb{Q})_\perp, \tag{84}$$

induce natural isomorphisms in **Lin** and we obtain a monoidal strength $\mathbb{P} \otimes \mathbb{Q}_\perp \to (\mathbb{P} \otimes \mathbb{Q})_\perp$ precisely as for **Cts**.

Finally, the monoidal closed structure of **Lin** together with the natural isomorphism $\mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P} \otimes \mathbb{Q})_\perp$ provide a right adjoint $(\mathbb{Q} \multimap -)$, defined by $(\mathbb{Q}_\perp \multimap -)$, to the functor $(- \otimes \mathbb{Q})$ in **Aff** via the chain

$$\begin{aligned} \mathbf{Aff}(\mathbb{P} \otimes \mathbb{Q}, \mathbb{R}) &\cong \mathbf{Lin}((\mathbb{P} \otimes \mathbb{Q})_\perp, \mathbb{R}) \cong \mathbf{Lin}(\mathbb{P}_\perp \times \mathbb{Q}_\perp, \mathbb{R}) \\ &\cong \mathbf{Lin}(\mathbb{P}_\perp, \mathbb{Q}_\perp \multimap \mathbb{R}) \cong \mathbf{Aff}(\mathbb{P}, \mathbb{Q}_\perp \multimap \mathbb{R}) = \mathbf{Aff}(\mathbb{P}, \mathbb{Q} \multimap \mathbb{R}), \end{aligned} \tag{85}$$

natural in $\mathbb{P}$ and $\mathbb{R}$. This demonstrates that **Aff** is symmetric monoidal closed and since the unit of the tensor is terminal, a model of affine linear logic, as already observed in [25].

5. Affine HOPLA

Affine HOPLA is a typed process language suggested by the structure of **Aff** [41]. Even though we replace the type constructor $!(-)$ by $(-)_\perp$, we will continue to use $!$ for the action in prefixing.

5.1. Denotational semantics

Types are given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \multimap \mathbb{T}_2 \mid \mathbb{T}_1 \otimes \mathbb{T}_2 \mid \sum_{\alpha\in A} \mathbb{T}_\alpha \mid \mathbb{T}_\perp \mid T \mid \mu_j \vec{T}.\vec{\mathbb{T}}. \tag{86}$$

Again, closed-type expressions are interpreted as path orders. For the solution of recursive-type definitions we proceed as for HOPLA, replacing finite sets of paths by sets P of size at most one, writing $\perp$ for the empty set:

$$p, q ::= P \mapsto q \mid P \otimes Q \mid \beta p \mid P \mid abs\ p. \tag{87}$$

Here, $P \otimes Q$ stands for a pair of paths P of $\mathbb{P}_\perp$ and Q of $\mathbb{Q}_\perp$ where at least one is non-$\perp$. Formation rules are displayed below alongside rules defining the ordering. Note that all path orders interpreting types of Affine HOPLA are posets because, unlike the exponential, the comonad $(-)_\perp$ maps posets to posets.

$$\begin{array}{rr}
\dfrac{P : \mathbb{P}_\perp \quad q : \mathbb{Q}}{P \mapsto q : \mathbb{P} \multimap \mathbb{Q}} & \dfrac{P' \leqslant_{\mathbb{P}_\perp} P \quad q \leqslant_{\mathbb{Q}} q'}{P \mapsto q \leqslant_{\mathbb{P}\multimap\mathbb{Q}} P' \mapsto q'} \\[2ex]
\dfrac{P : \mathbb{P}_\perp \quad Q : \mathbb{Q}_\perp \quad (P,Q) \neq (\perp,\perp)}{P \otimes Q : \mathbb{P} \otimes \mathbb{Q}} & \dfrac{P \leqslant_{\mathbb{P}_\perp} P' \quad Q \leqslant_{\mathbb{Q}_\perp} Q'}{P \otimes Q \leqslant_{\mathbb{P}\otimes\mathbb{Q}} P' \otimes Q'} \\[2ex]
\dfrac{p : \mathbb{P}_\beta \quad \beta \in A}{\beta p : \sum_{\alpha\in A} \mathbb{P}_\alpha} & \dfrac{p \leqslant_{\mathbb{P}_\beta} p' \quad \beta \in A}{\beta p \leqslant_{\sum_{\alpha\in A}\mathbb{P}_\alpha} \beta p'} \\[2ex]
\dfrac{}{\perp : \mathbb{P}_\perp} \quad \dfrac{p : \mathbb{P}}{\{p\} : \mathbb{P}_\perp} & \dfrac{P \leqslant_{\mathbb{P}} P'}{P \leqslant_{\mathbb{P}_\perp} P'} \\[2ex]
\dfrac{p : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{abs\ p : \mu_j \vec{T}.\vec{\mathbb{T}}} & \dfrac{p \leqslant_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} p'}{abs\ p \leqslant_{\mu_j\vec{T}.\vec{\mathbb{T}}} abs\ p'}
\end{array} \tag{88}$$

The raw syntax of terms is given by

$$\begin{aligned} t, u ::= {} & x \mid rec\, x.t \mid \sum_{i\in I} t_i \mid \lambda x.t \mid t\, u \mid t \otimes u \mid [u > x \otimes y \Rightarrow t] \mid \\ & \beta t \mid \pi_\beta t \mid !t \mid [u > !x \Rightarrow t] \mid abs\, t \mid rep\, t. \end{aligned} \tag{89}$$

The use of a pattern match term for tensor is similar to that in [2]. Let $\mathbb{P}_1, \ldots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and $x_1, \ldots, x_k$ distinct variables. A syntactic judgement $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ stands for a map

$$[\![x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]\!] : \mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k \to \mathbb{Q} \tag{90}$$

in **Aff**. When the environment list is empty, the corresponding tensor product is the empty path order $\mathbb{O}$. The term-formation rules for Affine HOPLA are very similar to those for HOPLA, replacing & by $\otimes$ in the handling of environment lists and the type constructors $!(-)$ and $\to$ by $(-)_\perp$ and $\multimap$. We discuss the remaining differences in the following.

New rules are introduced for the tensor operation:

$$\frac{\Gamma \vdash t : \mathbb{P} \quad \Lambda \vdash u : \mathbb{Q}}{\Gamma, \Lambda \vdash t \otimes u : \mathbb{P} \otimes \mathbb{Q}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Lambda \xrightarrow{u} \mathbb{Q}}{\Gamma \otimes \Lambda \xrightarrow{t \otimes u} \mathbb{P} \otimes \mathbb{Q}} \tag{91}$$

$$\frac{\Gamma, x : \mathbb{P}, y : \mathbb{Q} \vdash t : \mathbb{R} \quad \Lambda \vdash u : \mathbb{P} \otimes \mathbb{Q}}{\Gamma, \Lambda \vdash [u > x \otimes y \Rightarrow t] : \mathbb{R}} \qquad \frac{\Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R} \quad \Lambda \xrightarrow{u} \mathbb{P} \otimes \mathbb{Q}}{\Gamma \otimes \Lambda \xrightarrow{1_\Gamma \otimes u} \Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R}} \tag{92}$$

One important difference is the lack of contraction for the affine language. This restricts substitution of a common term into distinct variables, and so copying. The counterpart in the model is the absence of a suitable diagonal map from objects $\mathbb{P}$ to $\mathbb{P} \otimes \mathbb{P}$; for example, the map $X \mapsto X \otimes X$ from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{P} \otimes \mathbb{P}}$ is not in general a map in **Aff**.[5] Consider a term $t(x, y)$, with its free variables x and y shown explicitly, for which

$$x : \mathbb{P}, y : \mathbb{P} \vdash t(x, y) : \mathbb{Q}, \tag{93}$$

corresponding to a map $\mathbb{P} \otimes \mathbb{P} \xrightarrow{t} \mathbb{Q}$ in **Aff**. This does not generally entail that $x : \mathbb{P} \vdash t(x,x) : \mathbb{Q}$—there may not be a corresponding map in **Aff**, for example if $t(x, y) = x \otimes y$. Intuitively, if any computation for t involves both inputs, then $x : \mathbb{P} \vdash t(x,x) : \mathbb{Q}$ would use the same input twice and therefore cannot be interpreted in **Aff**. There is a syntactic condition on the occurrences of variables which ensures that in any computation, at most one of a set of variables is used.

Definition 5.1. Let v be a raw term. Say a set of variables V is *crossed in* v iff there are subterms of v of the form tensor $t \otimes u$, application $t\,u$, tensor match $[u > x \otimes y \Rightarrow t]$, or prefix match $[u > !x \Rightarrow t]$, for which v has free occurrences of variables from V appearing in both t and u.

If the set $\{x, y\}$ is *not* crossed in $t(x, y)$ above, then t uses at most one of its inputs x, y in each computation; semantically, t is interpreted as a map $\mathbb{P} \otimes \mathbb{P} \to \mathbb{Q}$ of **Aff** which behaves identically on input $X \otimes Y$ and $X \otimes \varnothing + \varnothing \otimes Y$ for all $X, Y \in \widehat{\mathbb{P}}$. In this case, $x : \mathbb{P} \vdash t(x,x) : \mathbb{Q}$ holds and is interpreted as the composition

$$\mathbb{P} \xrightarrow{\delta_{\mathbb{P}}} \mathbb{P} \otimes \mathbb{P} \xrightarrow{t} \mathbb{Q}, \tag{94}$$

[5] To see this, assume that $\mathbb{P}$ is the prefixed sum $\alpha.\mathbb{O} + \beta.\mathbb{O}$ with paths abbreviated to α, β. Confusing paths with the corresponding primes, the nonempty join $\alpha + \beta$ is sent by $X \mapsto X \otimes X$ to $\alpha \otimes \alpha + \beta \otimes \beta + \alpha \otimes \beta + \beta \otimes \alpha$ instead of $\alpha \otimes \alpha + \beta \otimes \beta$ as would be needed to preserve nonempty joins.

where $\delta_{\mathbb{P}} : \mathbb{P} \to \mathbb{P} \otimes \mathbb{P}$ maps X to $X \otimes \varnothing + \varnothing \otimes X$. We will write $\delta^k_{\mathbb{P}} : \mathbb{P} \to \mathbb{P}^k$ for the obvious generalisation to a k-fold tensor product $\mathbb{P}^k = \mathbb{P} \otimes \cdots \otimes \mathbb{P}$.

We can now give the rule for recursively defined processes in Affine HOPLA:

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{P} \quad \{x, y\} \text{ not crossed in } t \text{ for any } y \text{ in } \Gamma}{\Gamma \vdash rec\, x.t : \mathbb{P}} \qquad \frac{\Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}}{\Gamma \xrightarrow{fix\, F} \mathbb{P}} \tag{95}$$

Here, $fix\, F$ is the fixpoint in $\mathbf{Aff}(\Gamma, \mathbb{P}) \cong \widehat{\Gamma \multimap \mathbb{P}}$ of the continuous operation F mapping $g : \Gamma \to \mathbb{P}$ in $\mathbf{Aff}$ to the composition

$$\Gamma \xrightarrow{\delta_\Gamma} \Gamma \otimes \Gamma \xrightarrow{1_\Gamma \otimes g} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}. \tag{96}$$

5.2. Useful identities

Counterparts of the results for HOPLA of Section 3.2 can now be proved for Affine HOPLA. In particular, a general substitution lemma can be formulated as follows.

Lemma 5.2 (Substitution). *Suppose* $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P} \vdash t : \mathbb{Q}$ *with* $\{x_1, \ldots, x_k\}$ *not crossed in t. If* $\Lambda \vdash u : \mathbb{P}$ *with* Γ *and* Λ *disjoint, then* $\Gamma, \Lambda \vdash t[u/x_1, \ldots, u/x_k] : \mathbb{Q}$ *with denotation given by the composition*

$$\Gamma \otimes \Lambda \xrightarrow{1_\Gamma \otimes (\delta^k_{\mathbb{P}} \circ u)} \Gamma \otimes \mathbb{P}^k \xrightarrow{t} \mathbb{Q}. \tag{97}$$

An easy induction on typing derivations shows that if $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$, then $\{x\}$ is not crossed in t, and so the substitution lemma specialises to

Corollary 5.3. *If* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ *and* $\Lambda \vdash u : \mathbb{P}$ *with* Γ *and* Λ *disjoint, then we have* $\Gamma, \Lambda \vdash t[u/x] : \mathbb{Q}$ *with* $[\![(\lambda x.t)\, u]\!] = [\![t[u/x]]\!]$.

Corollary 5.4. *Suppose* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}$. *Then* $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ *with* $[\![rec\, x.t]\!] = [\![t[recx.t/x]]\!]$.

Proof. We proceed as in the proof of Corollary 3.3, obtaining $\Gamma', x : \mathbb{P} \vdash t' : \mathbb{P}$ *and* $\Gamma'', x : \mathbb{P} \vdash t'' : \mathbb{P}$ *with* Γ' *and* Γ'' *disjoint by renaming variables y of* Γ *to* y' *and* y''. *By the substitution lemma with* $k = 1$, *we get* $\Gamma', \Gamma'' \vdash t'[rec\, x.t''/x] : \mathbb{P}$ *denoting*

$$\Gamma' \otimes \Gamma'' \xrightarrow{1_{\Gamma'} \otimes rec\, x.t''} \Gamma' \otimes \mathbb{P} \xrightarrow{t'} \mathbb{P}. \tag{98}$$

Now, since the sets $\{x, y\}$ are not crossed in t, the sets $\{y', y''\}$ are not crossed in $t'[rec\, x.t''/x]$. Hence, by repeated use of exchange and the substitution lemma with $k = 2$, we may perform substitutions $[y/y', y/y'']$ to obtain $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ with denotation

$$\Gamma \xrightarrow{\delta_\Gamma} \Gamma \otimes \Gamma \xrightarrow{1_\Gamma \otimes rec\, x.t} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}. \tag{99}$$

Again, this is the same as $F(\mathit{fix}\, F) = \mathit{fix}\, F$, the denotation of $\mathit{rec}\, x.t$. □

The properties of sums and prefixing are the same as for HOPLA.

Proposition 5.5. *The tensor match satisfies*

$$[\![[u_1 \otimes u_2 > x \otimes y \Rightarrow t]]\!] = [\![t[u_1/x, u_2/y]]\!], \tag{100}$$

$$[\![[u > x \otimes y \Rightarrow \sum_{i\in I} t_i]]\!] = [\![\sum_{i\in I} [u > x \otimes y \Rightarrow t_i]]\!], \tag{101}$$

$$[\![[\sum_{i\in I} u_i > x \otimes y \Rightarrow t]]\!] = [\![\sum_{i\in I} [u_i > x \otimes y \Rightarrow t]]\!] \quad \textit{if } I \neq \varnothing. \tag{102}$$

Further, if x_1 and y_1 are not free in t, then

$$\begin{aligned} &[\![[[u_1 > x_1 \otimes y_1 \Rightarrow u_2] > x_2 \otimes y_2 \Rightarrow t]]\!] \\ &\quad = [\![[u_1 > x_1 \otimes y_1 \Rightarrow [u_2 > x_2 \otimes y_2 \Rightarrow t]]]\!]. \end{aligned} \tag{103}$$

Proof. All the properties are consequences of tensor match being interpreted as composition in **Aff**. Eq. (100) follows by exchange and two applications of the substitution lemma. The two distributive properties hold since composition $f \circ g$ in **Aff** is linear in f and affine in g. Finally, (103) follows from associativity of composition. □

5.3. Full abstraction

As for HOPLA, we take a program to be a closed term t of type $\mathbb{O}_\perp$, but because of linearity constraints, program contexts will now have at most one hole. Otherwise, the notion of contextual preorder is the same as in Section 3.3. Again, contextual equivalence coincides with path equivalence:

Theorem 5.6 (Full abstraction). *For any terms $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$,*

$$[\![t_1]\!] \subseteq [\![t_2]\!] \;\Leftrightarrow\; t_1 \sqsubseteq_\sim t_2. \tag{104}$$

Proof. Path "realisers" and "consumers" are defined as in the proof of full abstraction for HOPLA, restricting the terms t'_P and C'_P to the cases where P has at most one element. Terms corresponding to paths of tensor type are defined by

$$\begin{aligned} t_{P\otimes Q} &\equiv t'_P \otimes t'_Q, \\ C_{P\otimes Q} &\equiv [- > x \otimes y \Rightarrow [C'_P(x) > !x' \Rightarrow C'_Q(y)]]. \end{aligned} \tag{105}$$

For any $p : \mathbb{P}$ and $P : \mathbb{P}$ we then have (z being a fresh variable)

$$\begin{aligned} &[\![t_p]\!] = \mathrm{y}_{\mathbb{P}}\, p, \quad [\![\lambda z.C_p(z)]\!] = \mathrm{y}_{\mathbb{P}\multimap\mathbb{O}_\perp}(\{p\} \mapsto \varnothing), \\ &[\![t'_P]\!] = j_{\mathbb{P}} P, \quad [\![\lambda z.C'_P(z)]\!] = \mathrm{y}_{\mathbb{P}\multimap\mathbb{O}_\perp}(P \mapsto \varnothing). \end{aligned} \tag{106}$$

We can now proceed as in the proof of Theorem 3.7. □

5.4. Expressive power

Subject to the linearity constraints on occurrences of variables, Affine HOPLA has much of the expressive power of HOPLA. In particular, the calculi discussed in Sections 3.6.2 and 3.6.3 can be encoded with the restriction that no variable can occur freely on both sides of a parallel composition. The prefixed sum $\sum_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ stands for $\sum_{\alpha\in A}(\mathbb{P}_\alpha)_\perp$ in Affine HOPLA. Prefixing $\beta.t$ is still translated into $\beta!t$, but now has a different semantics. For example, by replacing $!(-)$ with $(-)_\perp$, the solution of Eq. (71) defining the type of CCS processes becomes isomorphic to the partial order of strings over the alphabet of CCS actions. Thus, the semantics of CCS given by the translation into Affine HOPLA is a traditional trace semantics. This is illustrated by the fact that the two CCS processes $\alpha.\beta.\varnothing+\alpha.\gamma.\varnothing$ and $\alpha.(\beta.\varnothing+\gamma.\varnothing)$ are given the same semantics by the Affine HOPLA translation, but can be told apart by the HOPLA context $C_{\langle\alpha\rangle(\langle\beta\rangle\top\wedge\langle\gamma\rangle\top)}$, see Section 3.5.

More interestingly, the tensor type of Affine HOPLA allows us to define processes of the kind encountered in treatments of nondeterministic dataflow [27], something which is not possible using HOPLA. To illustrate, define $\mathbb{P}$ recursively as the prefixed sum

$$\mathbb{P}=\alpha.\mathbb{P}+\beta.\mathbb{P}, \tag{107}$$

so that $\mathbb{P}$ essentially consists of streams (or sequences) of α's and β's. We can then define dataflow processes whose properties can be determined from the above results about the denotational semantics—in particular using Proposition 5.5:

- A process A of type $\mathbb{P}\otimes\mathbb{P}$ which produces two identical, parallel streams of α's and β's as output:

$$A\equiv rec\,p.[p>x\otimes y\Rightarrow(\alpha.x\otimes\alpha.y)+(\beta.x\otimes\beta.y)]. \tag{108}$$

  The denotation of A is the set of pairs (s,s') with s and s' strings of α's and β's, such that s is a prefix of s' or vice versa. Notice the "entanglement" between the two sides of the tensor—choices made on one side affect choice on the other.
- A process B of type $\mathbb{P}\multimap(\mathbb{P}\otimes\mathbb{P})$ which is like A, except it produces its two output streams as copies of the input stream:

$$\begin{aligned}B\equiv rec\,f.\lambda z.[z>\alpha.z'\Rightarrow[f\,z'>x\otimes y\Rightarrow\alpha.x\otimes\alpha.y]]\\+[z>\beta.z'\Rightarrow[fz'>x\otimes y\Rightarrow\beta.x\otimes\beta.y]].\end{aligned} \tag{109}$$

  We have e.g. $[\![B(\alpha.\beta.\varnothing)]\!]=[\![\alpha.\beta.\varnothing\otimes\alpha.\beta.\varnothing]\!]$ and $[\![B(\alpha.\varnothing+\beta.\varnothing)]\!]=[\![\alpha.\varnothing\otimes\alpha.\varnothing+\beta.\varnothing\otimes\beta.\varnothing]\!]$, the latter *not* containing "cross terms" like $\alpha.\varnothing\otimes\beta.\varnothing$.
- A process C of type $(\mathbb{P}\otimes\mathbb{P})\multimap\mathbb{P}$ which merges two streams into one:

$$\begin{aligned}C\equiv rec\,f.\lambda z.[z>x\otimes y\Rightarrow[x>\alpha.x'\Rightarrow\alpha.f(y\otimes x')]\\+[x>\beta.x'\Rightarrow\beta.f(y\otimes x')]].\end{aligned} \tag{110}$$

We have e.g. $[\![C(\alpha.\alpha.\varnothing\otimes\beta.\beta.\varnothing)]\!]=[\![\alpha.\beta.\alpha.\beta.\varnothing]\!]$.

A "trace operation" to represent dataflow processes with feedback loops is not definable in Affine HOPLA, because then we would have obtained a compositional relational

semantics of nondeterministic dataflow with feedback, shown impossible by Brock and Ackerman [9]. However, with a more refined notion of "relation", which spells out the different ways in which input and output of a dataflow process are related, such a semantics is in fact possible [23].

6. Related work

We conclude by setting the specific results of this paper in the context of what we see as a promising broader enterprise towards a full domain theory for concurrency.

6.1. Presheaf semantics

We have investigated the path semantics of HOPLA and Affine HOPLA. In reality, HOPLA and Affine HOPLA were discovered within a more informative domain theory than that based on path sets. As remarked earlier, the domain of path sets $\widehat{\mathbb{P}}$, of a path order $\mathbb{P}$, is isomorphic to characteristic functions $[\mathbb{P}^{\mathrm{op}}, \mathbf{2}]$, ordered pointwise. In modelling a process as a path set, we are in effect representing a process by a characteristic function from paths to truth values $0 < 1$. If instead of these simple truth values we take sets of realisers, replacing $\mathbf{2}$ by the category of sets **Set**, we obtain a functor category $[\mathbb{P}^{\mathrm{op}}, \mathbf{Set}]$, whose objects, traditionally called presheaves, provide an alternative "domain" of meanings; now a process denotes a presheaf in which a path is associated with the set of elements standing for the ways in which the path can be realised.

For the presheaf semantics of HOPLA we can obtain a more refined adequacy result than that for the path semantics: Letting $\vdash t : !\mathbb{P}$, the set of realisers $[\![t]\!](\varnothing)$ corresponds to the set of derivations of $!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P}$. In fact, a guiding principle in designing the operational semantics has been that derivations of transitions of which the actions are essentially paths should correspond to the realisers associated to the path in the denotational semantics; this generally determines the form of rules.

A presheaf captures the nondeterministic branching of a process and a presheaf semantics can support equivalences such as forms of bisimulation which are sensitive to the branching behaviour of processes. Though here our understanding of the role of open maps and open map bisimulation, intrinsic to presheaf models [26], is very incomplete.

The presheaf semantics helps expose a range of possible pseudo comonads with which to interpret $!\mathbb{P}$ [15,41].

6.2. Powerdomains

The adjunction between **Lin** and **Cts**, key to our semantics of HOPLA, determines a monad, the monad of the "Hoare powerdomain" [52]. The adjunction between **Lin** and **Cts** is of the kind already studied in the early work of Hennessy and Plotkin [20]; they were concerned with adjunctions between categories of nondeterministic cpos and categories of cpos associated with a variety of powerdomains. This was in the days

prior to linear logic. But models of linear logic are obtained by cutting down their adjunctions.

Like the model of linear logic formed from **Lin** and **Cts**, we expect that each model furnishes a denotational semantics of HOPLA. Presumably there are full abstraction results companion to that here based on detecting the "must" as well as "may" behaviour of processes. Just as there is an abstraction function from the presheaf semantics of HOPLA to its path semantics (induced by sending nonempty sets of realisers to 1 and the empty set to 0), so can we expect other abstraction functions from the presheaf semantics to other powerdomain semantics. But presently all this is conjectural.

Note that this use of powerdomains does not fit the original pattern proposed for handling concurrency via a recursively defined powerdomain of resumptions [46]; rather one defines domains of paths recursively and only then adjoins nondeterminism.

6.3. An underlying language?

Most process languages have developed incrementally, based on previously known languages. Even HOPLA and Affine HOPLA are essentially lambda-calculi extended by nondeterministic sum and prefix operations (though the latter are understood as arising from a comonad associated with models of linear logic). Proof theory is beginning to influence ideas on the nature of processes. A recent impetus has been the discovery of linear logic, a discovery founded on the domain theory of coherence spaces with linear and stable maps [19]. Similarly, we can hope that a persuasive mathematical model of processes can guide us towards a fuller understanding of processes and their syntax.

We have a rich model in the linear category analogous to **Lin** but based on presheaves rather than path sets. Just as maps in **Lin** correspond to relations, the analogous maps correspond to profunctors, a generalisation of relations (see e.g. [7], for an elementary introduction to profunctors, there called "distributors"). The bicategory of profunctors **Prof** is analogous to **Lin**.[6] Like **Lin** the bicategory **Prof** has an involution so that maps $f : \mathbb{P} \to \mathbb{Q}$ correspond to their dual $f^{\perp} : \mathbb{Q}^{\perp} \to \mathbb{P}^{\perp}$. Indeed, again just as in **Lin**, a map $f : \mathbb{P} \to \mathbb{Q}$ corresponds to a map $f' : \mathbb{P} \times \mathbb{Q}^{\perp} \to \mathbb{1}$, in which we have "dualised" the output to input.

It is because of this duality that open maps and open-map bisimulation for higher-order processes take as much account of input as they do output. Most often two higher-order processes are defined to be bisimilar iff they yield bisimilar outputs on any common input. But this simply won't do within a type discipline in which all nontrivial output can be "dualised" to input. On the other hand, traditional process languages and their types don't support this duality.

One line towards understanding open-map bisimulation at higher order is to design a process language in which this duality is present. The language could support the types

[6] The bicategory **Prof** is equivalent to the 2-category in which maps are colimit-preserving functors between presheaf categories, perhaps a more immediate analogue of **Lin**.

of **Prof** extended by a suitable pseudo comonad. Ideally one would obtain a coinductive charactisation of open map bisimulation at higher order based on an operational semantics. (The mathematics for this enterprise is developed in [15].)

6.4. Affine models

Linear maps alone are too restrictive to support a semantics of processes. To do so, they must be moderated through the use of a (pseudo) comonad, the simplest of which is lifting.

There is a category analogous to **Aff** based on presheaves rather than path sets; its maps preserve connected colimits in presheaf categories [15,41]. This affine category is host to the semantics of nondeterministic dataflow [23], event-structure semantics of CCS and related languages [14] as well as a semantics for Affine HOPLA.

It came as a recent surprise [40] that the presheaf denotations of first-order processes in Affine HOPLA can be represented by event structures; the elements of definable presheaves can be understood as finite configurations of an event structure. In more detail, maps definable in Affine HOPLA by open terms can be represented by certain spans of event structures with composition given by pullbacks. This sheds light on the tensor operation and the form of entanglement associated with it, revealing the tensor as a form of parallel composition of event structures and entanglement as a pattern of concurrency/conflict. The event-structure semantics extends to all types, so higher-order processes. Though, as one would expect, the event-structure semantics diverges from the presheaf semantics at higher-order; the event-structure semantics is analogous to stable domain theory [6].

As mentioned above, we can define a semantics for CCS using Affine HOPLA subject to certain restrictions on occurrences of variables. Unfortunately, one can show the event-structure denotations of Affine HOPLA are too impoverished to coincide with the standard "true concurrency" semantics of CCS as, e.g. given in [55]. A language must go beyond Affine HOPLA if it is to express such semantics. Guidelines on what's lacking in Affine HOPLA can be got from work on presheaf models for concurrency [12], where the ingredients of product of presheaves, pomset augmentation and cartesian liftings (extending the match operators of Affine HOPLA) all play a critical role. This work suggests exploring other event-structure representations, based on more general spans of event structures, and perhaps a new comonad yielding a less rigid form of prefixing.

As a general point, the affine category based on presheaves is very rich in structure and supports a great many mathematical constructions which lie outside the scope of the present syntax of Affine HOPLA.

An operational semantics for the tensor-fragment of Affine HOPLA (leaving out function space) was given in [41]. But it has proved very challenging to extend this to higher order. Linearity obliges us to work with rather complicated environments, and entanglement of terms of tensor type in the execution of processes. (Note that the simplifying equation (102) is *not* valid in the presheaf semantics, not even up to isomorphism, because there, affine maps preserve *connected* colimits, and any nontrivial sum is manifestly not connected.) It is the interaction of the environments with

higher-order processes which has been problematic in giving an operational semantics to full Affine HOPLA.

However the event-structure denotational semantics of Affine HOPLA suggests an alternative operational semantics obviating the need for complicated environments. It is at the cost of having transitions between *open* terms. Taking advantage of stability, the configurations of an event structure representing an open term $x : \mathbb{P} \vdash t : \mathbb{Q}$, will be associated with both an output $q \in \mathbb{Q}$ and a minimal input, $P \in \mathbb{P}_\perp$ necessary for that output. The idea is that such a configuration will correspond to a derivation in the operational semantics of a transition $x : P \vdash t \xrightarrow{q} t'$ [40].

6.5. Name generation

Process languages often follow the pioneering work on the π-calculus and allow name generation. HOPLA can be extended to encompass such languages [54]. The extensions are to add a type of names $\mathcal{N}$, function spaces, as well as a type $\delta\mathbb{P}$ supporting new-name generation through the abstraction *new x.t*. The denotational semantics of the extension to name generation is currently being developed; this addresses the question of when function spaces exist in the obvious model (extending that of [13]). There is already an operational semantics; it is like that of HOPLA but given at stages indexed by the current set of names.

Acknowledgements

The authors are glad to take this opportunity, that of Dana Scott's retirement, to express their indebtedness to Dana Scott for his pioneering and visionary work in logic and computation. GW would like to add his gratitude for many past kindnesses.

Appendix A. Proof of soundness

We want to show that if $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$, then $[\![!t']\!] \subseteq a^*[\![t]\!]$. The proof is by rule-induction on the transition rules:

Recursive definition: If $\mathbb{P} : rec\, x.t \xrightarrow{a} t' : \mathbb{P}'$ we have $\mathbb{P} : t[rec\, x.t/x] \xrightarrow{a} t' : \mathbb{P}'$ as premise. By the induction hypothesis and Corollary 3.3,

$$[\![!t']\!] \subseteq a^*[\![t[rec\, x.t/x]]\!] = a^*[\![rec\, x.t]\!]. \tag{A.1}$$

Nondeterministic sum: If $\mathbb{P} : \sum_{i\in I} t_i \xrightarrow{a} t' : \mathbb{P}'$ we have the premise $\mathbb{P} : t_j \xrightarrow{a} t' : \mathbb{P}'$ for some $j \in I$. By the induction hypothesis and linearity of a^*,

$$[\![!t']\!] \subseteq a^*[\![t_j]\!] = [\![a^* t_j]\!] \subseteq [\![\sum_{i\in I} a^* t_i]\!] = a^*[\![\sum_{i\in I} t_i]\!]. \tag{A.2}$$

Abstraction: If $\mathbb{P} \to \mathbb{Q} : \lambda x.t \overset{u \mapsto a}{\longrightarrow} t' : \mathbb{P}'$ we have $\mathbb{Q} : t[u/x] \xrightarrow{a} t' : \mathbb{P}'$ as premise. By the induction hypothesis and Corollary 3.2,

$$[\![!t']\!] \subseteq a^*[\![t[u/x]]\!] = a^*[\![(\lambda x.t)\, u]\!] = (u \mapsto a)^*[\![\lambda x.t]\!]. \tag{A.3}$$

Application: If $\mathbb{Q} : t\,u \xrightarrow{a} t' : \mathbb{P}'$ we have the premise $\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t' : \mathbb{P}'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (u \mapsto a)^*[\![t]\!] = a^*[\![t\,u]\!]. \tag{A.4}$$

Injection: If $\sum_{\alpha\in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta a} t' : \mathbb{P}'$ we have the premise $\mathbb{P}_\beta : t \xrightarrow{a} t'' : \mathbb{P}'$. By the induction hypothesis and Proposition 3.4,

$$[\![!t']\!] \subseteq a^*[\![t]\!] = a^*[\![\pi_\beta(\beta t)]\!] = (\beta a)^*[\![\beta t]\!]. \tag{A.5}$$

Projection: If $\mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t' : \mathbb{P}'$ we have the premise $\sum_{\alpha\in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t' : \mathbb{P}'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (\beta a)^*[\![t]\!] = a^*[\![\pi_\beta t]\!]. \tag{A.6}$$

Prefixing: Consider the transition $!\mathbb{P} : !t \xrightarrow{!} t : \mathbb{P}$. By definition, $!^*[\![!t]\!] = [\![!t]\!]$, a subset of itself.

Prefix match: If $\mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{a} t' : \mathbb{P}'$ we have the premises $!\mathbb{P} : u \xrightarrow{!} u' : \mathbb{P}$ and $\mathbb{Q} : t[u'/x] \xrightarrow{a} t' : \mathbb{P}'$. By the induction hypothesis for u,

$$[\![!u']\!] \subseteq !^*[\![u]\!] = [\![u]\!]. \tag{A.7}$$

Now by the induction hypothesis for t, Proposition 3.5 and monotonicity,

$$[\![!t']\!] \subseteq a^*[\![t[u'/x]]\!] = a^*[\![[!u' > !x \Rightarrow t]]\!] \subseteq a^*[\![u > !x \Rightarrow t]]\!]. \tag{A.8}$$

Fold: If $\mu_j \vec{T}.\vec{\mathbb{T}} : abs\,t \xrightarrow{abs\,a} t' : \mathbb{P}'$, we have the premise $\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{a} t' : \mathbb{P}'$. By the induction hypothesis and since *abs* and *rep* are inverses,

$$[\![!t']\!] \subseteq a^*[\![t]\!] = a^*[\![rep(abs\,t)]\!] = (abs\,a)^*[\![abs\,t]\!]. \tag{A.9}$$

Unfold: If $\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\,t \xrightarrow{a} t' : \mathbb{P}'$, we have the premise $\mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{abs\,a} t' : \mathbb{P}'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (abs\,a)^*[\![t]\!] = a^*[\![rep\,t]\!]. \tag{A.10}$$

The rule-induction is complete.

Appendix B. Proof of adequacy (main lemma)

For the proof of Lemma 3.11 we need two technical results, which can both be proved by induction on the structure of paths. One says that $\varepsilon_{\mathbb{P}}$ is closed on the left by $\leqslant_{\mathbb{P}}$, the other that $\varepsilon_{\mathbb{P}}$ is closed on the right by the relation $\sqsubseteq_1$, defined by $t_1 \sqsubseteq_1 t_2$ iff $\mathbb{P} : t_1 \xrightarrow{a} t' : \mathbb{P}'$ implies $\mathbb{P} : t_2 \xrightarrow{a} t' : \mathbb{P}'$.

Lemma B.1. *If* $p \leqslant_{\mathbb{P}} p'$ *and* $p' \,\varepsilon_{\mathbb{P}}\, t$, *then* $p \,\varepsilon_{\mathbb{P}}\, t$.

Lemma B.2. *If* $p \,\varepsilon_{\mathbb{P}}\, t_1$ *and* $t_1 \sqsubseteq_1 t_2$, *then* $p \,\varepsilon_{\mathbb{P}}\, t_2$.

It follows from Lemma B.1 that for any subset X of $\mathbb{P}$ we have $X \trianglelefteq_{\mathbb{P}} t$ iff the down-closure of X, written $\bar{X}$, satisfies $\bar{X} \trianglelefteq_{\mathbb{P}} t$.

The proof of Lemma 3.11 proceeds by structural induction on terms using the induction hypothesis

Suppose $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{P}$ and let $\vdash s_j : \mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. Then $[\![t]\!](\overline{X_1}, \ldots, \overline{X_k}) \trianglelefteq_{\mathbb{P}} t[s_1/x_1, \ldots, s_k/x_k]$.

We will abbreviate $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$ to Γ, $(\overline{X_1}, \ldots, \overline{X_k})$ to X, and the substitution $[s_1/x_1, \ldots, s_k/x_k]$ to $[s]$. Lemma B.2 will be used freely below.

Variable: Let $\Gamma \vdash x_j : \mathbb{P}_j$, with j between 1 and k, and $\vdash s_j : \mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![x_j]\!]X \trianglelefteq_{\mathbb{P}_j} x_j[s]$. Now, $[\![x_j]\!]X = \overline{X_j}$ and $x_j[s] \equiv s_j$ so this amounts to $\overline{X_j} \trianglelefteq_{\mathbb{P}_j} s_j$ which by the remarks above is equivalent to $X_j \trianglelefteq_{\mathbb{P}_j} s_j$.

Recursive definition: Let $\Gamma \vdash rec\, x.t : \mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![rec\, x.t]\!]X \trianglelefteq_{\mathbb{P}} rec\, x.t[s]$. Now, $[\![rec\, x.t]\!]X = (fix\, F)X$ where F maps $g : \Gamma \to \mathbb{P}$ to the composition

$$\Gamma \xrightarrow{\Delta_\Gamma} \Gamma \& \Gamma \xrightarrow{1_\Gamma \& g} \Gamma \& \mathbb{P} \xrightarrow{t} \mathbb{P}. \tag{B.1}$$

We will show by induction on n that $F^n(\varnothing)X \trianglelefteq_{\mathbb{P}} rec\, x.t[s]$ for all $n \in \omega$. Having done so we may argue as follows: Since

$$[\![rec\, x.t]\!]X = (fix\, F)X = \left(\bigcup_{n\in\omega} F^n\varnothing\right)X = \bigcup_{n\in\omega} ((F^n\varnothing)X), \tag{B.2}$$

we have that $p \in [\![rec\, x.t]\!]X$ implies the existence of an $n \in \omega$ such that $p \in (F^n\varnothing)X$. Therefore $[\![rec\, x.t]\!]X \trianglelefteq_{\mathbb{P}} rec\, x.t[s]$ as wanted.

Basis: Here, $(F^0\varnothing)X = \varnothing$. By definition of $\trianglelefteq_{\mathbb{P}}$ we get $\varnothing \trianglelefteq_{\mathbb{P}} t$ for any type $\mathbb{P}$ and term $\vdash t : \mathbb{P}$.

Step: Suppose $(F^n\varnothing)X \trianglelefteq_{\mathbb{P}} rec\, x.t[s]$. By the assumption of the lemma, $X_j \trianglelefteq_{\mathbb{P}} s_j$ for each $1 \leqslant j \leqslant k$, and so by the induction hypothesis of the structural induction,

$$[\![t]\!](X, (F^n\varnothing)X) \trianglelefteq_{\mathbb{P}} t[s][rec\, x.t[s]/x]. \tag{B.3}$$

So if $p \in (F^{n+1}\varnothing)X$, then since $(F^{n+1}\varnothing)X = [\![t]\!](X, (F^n\varnothing)X)$ we have $p \,\varepsilon_{\mathbb{P}}\, t[s][recx.t[s]/x]$. By the transition rules we have $t[s][rec\, x.t[s]/x] \sqsubseteq_1 \lesssim_1 rec\, x.t[s]$, and so $p \,\varepsilon_{\mathbb{P}}\, rec\, x.t\,[s]$. We conclude $(F^{n+1}\varnothing)X \trianglelefteq_{\mathbb{P}} rec\, x.t[s]$ and the mathematical induction is complete.

Nondeterministic sum: Let $\Gamma \vdash \sum_{i\in I} t_i : \mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![\sum_{i\in I} t_i]\!]X \trianglelefteq_{\mathbb{P}} \sum_{i\in I} t_i[s]$. Now, $[\![\sum_{i\in I} t_i]\!]X = \sum_{i\in I} [\![t_i]\!]X$. So if $p \in [\![\sum_{i\in I} t_i]\!]X$, there exists $j \in I$ with $p \in [\![t_j]\!]X$. Using the induction hypothesis for t_j we have $p \,\varepsilon_{\mathbb{P}}\, t_j[s]$. By the transition rules, $t_j[s] \sqsubseteq_1 \sum_{i\in I} t_i[s]$ and so $p \,\varepsilon_{\mathbb{P}}\, \sum_{i\in I} t_i[s]$ as wanted.

Abstraction: Let $\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![\lambda x.t]\!]X \trianglelefteq_{\mathbb{P}\to\mathbb{Q}} (\lambda x.t)[s]$. So let $P \mapsto q \in [\![\lambda x.t]\!]X$. By the denotational semantics, we then have $q \in [\![t]\!](X, i_{\mathbb{P}}P)$. We must show that $P \mapsto q \,\varepsilon_{\mathbb{P}\to\mathbb{Q}}\, (\lambda x.t)[s]$. So

suppose $\vdash u:\mathbb{P}$ with $P \trianglelefteq_{\mathbb{P}} u$. We must then show $q\,\varepsilon_{\mathbb{Q}}\,(\lambda x.t)[s]\,u$. By the transition rules, $t[s][u/x] \sqsubseteq_1 (\lambda x.t)[s]\,u$ and so it is sufficient to show $q\,\varepsilon_{\mathbb{Q}}\,t[s][u/x]$. Now, by the induction hypothesis, we know that $[\![t]\!](X, i_{\mathbb{P}}P) \trianglelefteq_{\mathbb{Q}} t[s][u/x]$ and so, with $q \in [\![t]\!](X, i_{\mathbb{P}}P)$, we are done.

Application: Let $\Gamma \vdash t\,u:\mathbb{Q}$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![t\,u]\!]X \trianglelefteq_{\mathbb{Q}} (t\,u)[s]$. So suppose $q \in [\![t\,u]\!]X$. By the denotational semantics, there exists $P \in !\mathbb{P}$ such that $P \mapsto q \in [\![t]\!]X$ and $P \subseteq [\![u]\!]X$. By the induction hypothesis for t, we have $[\![t]\!]X \trianglelefteq_{\mathbb{P}\to\mathbb{Q}} t[s]$ and so $P \mapsto q\,\varepsilon_{\mathbb{P}\to\mathbb{Q}}\,t[s]$. This means that given any $\vdash u':\mathbb{P}$ with $P \trianglelefteq_{\mathbb{P}} u'$, we have $q\,\varepsilon_{\mathbb{Q}}\,t[s]\,u'$. Now using the induction hypothesis for u we get that $[\![u]\!]X \trianglelefteq_{\mathbb{P}} u[s]$ and so, since $P \subseteq [\![u]\!]X$, we have $P \trianglelefteq_{\mathbb{P}} u[s]$ so that $q\,\varepsilon_{\mathbb{Q}}\,t[s]\,u[s] \equiv (t\,u)[s]$ as wanted.

Injection: Let $\Gamma \vdash \beta t:\sum_{\alpha\in A}\mathbb{P}_\alpha$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![\beta t]\!]X \trianglelefteq_{\sum_{\alpha\in A}\mathbb{P}_\alpha} (\beta t)[s]$. So suppose $\beta\,p \in [\![\beta t]\!]X$; by the denotational semantics, $p \in [\![t]\!]X$. We must then show that $\beta\,p\,\varepsilon_{\sum_{\alpha\in A}\mathbb{P}_\alpha}(\beta t)[s]$ which means that $p\,\varepsilon_{\mathbb{P}_\beta}\,\pi_\beta(\beta t[s])$. By the transition rules, we have $t[s] \sqsubseteq_1 \pi_\beta(\beta t[s])$ so it is sufficient to show that $p\,\varepsilon_{\mathbb{P}_\beta}\,t[s]$. By the induction hypothesis, $[\![t]\!]X \trianglelefteq_{\mathbb{P}_\beta} t[s]$ and so, since $p \in [\![t]\!]X$ we have $p\,\varepsilon_{\mathbb{P}_\beta}\,t[s]$ as wanted.

Projection: Let $\Gamma \vdash \pi_\beta t:\mathbb{P}_\beta$ with $\Gamma \vdash t:\sum_{\alpha\in A}\mathbb{P}_\alpha$ and $\beta \in A$, and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![\pi_\beta t]\!]X \trianglelefteq_{\mathbb{P}_\beta} \pi_\beta t[s]$. So suppose $p \in [\![\pi_\beta t]\!]X$; by the denotational semantics, $\beta\,p \in [\![t]\!]X$. By the induction hypothesis, $[\![t]\!]X \trianglelefteq_{\sum_{\alpha\in A}\mathbb{P}_\alpha} t[s]$ and so $\beta\,p\,\varepsilon_{\sum_{\alpha\in A}\mathbb{P}_\alpha} t[s]$ which means that $p\,\varepsilon_{\mathbb{P}_\beta}\,\pi_\beta t[s]$ as wanted.

Prefixing: Let $\Gamma \vdash !t:!\mathbb{P}$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![!t]\!]X \trianglelefteq_{!\mathbb{P}} !t[s]$. So suppose $P \in [\![!t]\!]X$; by the denotational semantics, $P \subseteq [\![t]\!]X$. We must then show that $P\,\varepsilon_{!\mathbb{P}}\,!t[s]$, and so since the transition rules provide a derivation $!\mathbb{P}:!t[s] \xrightarrow{!} t[s]:\mathbb{P}$, that $P \trianglelefteq_{\mathbb{P}} t[s]$. Now, by the induction hypothesis, $[\![t]\!]X \trianglelefteq_{\mathbb{P}} t[s]$ and so, since $P \subseteq [\![t]\!]X$ we have $P \trianglelefteq_{\mathbb{P}} t[s]$ as wanted.

Prefix match: Let $\Gamma \vdash [u > !x \Rightarrow t]:\mathbb{Q}$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. By renaming x if necessary, we may assume that x is not one of the x_j. We must show that $[\![[u > !x \Rightarrow t]]\!]X \trianglelefteq_{\mathbb{Q}} [u > !x \Rightarrow t][s]$. So suppose $q \in [\![[u > !x \Rightarrow t]]\!]X$; by the denotational semantics, there exists $P \in !\mathbb{P}$ such that $q \in [\![t]\!](X, i_{\mathbb{P}}P)$ and $P \in [\![u]\!]X$. By the induction hypothesis for u we have $[\![u]\!]X \trianglelefteq_{!\mathbb{P}} u[s]$ and so since $P \in [\![u]\!]X$, there exists u' such that $!\mathbb{P}:u[s] \xrightarrow{!} u':\mathbb{P}$ and $P \trianglelefteq_{\mathbb{P}} u'$. Hence, by the induction hypothesis for t we have $[\![t]\!](X, i_{\mathbb{P}}P) \trianglelefteq_{\mathbb{Q}} t[s][u'/x]$ and so since $q \in [\![t]\!](X, i_{\mathbb{P}}P)$ we have $q\,\varepsilon_{\mathbb{Q}}\,t[s][u'/x]$. Now, by the transition rules, $t[s][u'/x] \sqsubseteq_1 [u > !x \Rightarrow t][s]$ and so $q\,\varepsilon_{\mathbb{Q}}\,[u > !x \Rightarrow t][s]$ as wanted.

Fold: Let $\Gamma \vdash abs\,t:\mu_j\vec{T}.\vec{\mathbb{T}}$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![abs\,t]\!]X \trianglelefteq_{\mu_j\vec{T}.\vec{\mathbb{T}}} abs\,t[s]$. So suppose $abs\,q \in [\![abs\,t]\!]X$ such that $q \in [\![t]\!]X$. By the induction hypothesis, $q\,\varepsilon_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}\,t[s]$ and since $t[s] \sqsubseteq_1 rep\,abs\,t[s]$, we have $q\,\varepsilon_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}\,rep\,abs\,t[s]$ which means that $abs\,q\,\varepsilon_{\mu_j\vec{P}.\vec{\mathbb{T}}}\,abs\,t[s]$ as wanted.

Unfold: Let $\Gamma \vdash rep\,t:\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]$ and $\vdash s_j:\mathbb{P}_j$ with $X_j \trianglelefteq_{\mathbb{P}_j} s_j$ for $1 \leqslant j \leqslant k$. We must show that $[\![rep\,t]\!]X \trianglelefteq_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\,t[s]$. So suppose $q \in [\![rep\,t]\!]X$ such that $abs\,q \in [\![t]\!]X$. By the induction hypothesis, $abs\,q\,\varepsilon_{\mu_j\vec{T}.\vec{\mathbb{T}}}\,t[s]$ and so $q\,\varepsilon_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}\,rep\,t[s]$ as wanted.

The structural induction is complete.

References

[1] S. Abramsky, The lazy lambda calculus, in: D. Turner (Ed.), Research Topics in Functional Programming, Addison-Wesley, Reading, MA, 1990.

[2] S. Abramsky, Computational interpretations of linear logic, Theoret. Comput. Sci. 111 (1–2) (1993) 3–57.

[3] S. Abramsky, Game semantics for programming languages, in: Proc. MFCS'97, Lecture Notes in Computer Science, Vol. 1295, Springer, Berlin, 1997, pp. 3–4.

[4] P.N. Benton, A mixed linear and non-linear logic: proofs, terms and models (extended abstract), in: Proc. CSL'94, Lecture Notes in Computer Science, Vol. 933, Springer, Berlin, 1994, pp. 121–135.

[5] P.N. Benton, G. Bierman, V. de Paiva, M. Hyland, Linear λ-calculus and categorical models revisited, in: Proc. CSL'92, Lecture Notes in Computer Science, Vol. 702, Springer, Berlin, 1992, pp. 61–84.

[6] G. Berry, Modèles complètement adéquats et stables des lambda-calculs typés, Thèse de Doctorat d'Etat, Université Paris VII, 1979.

[7] F. Borceux, Handbook of Categorical Algebra I. Basic Category Theory, Cambridge University Press, Cambridge, 1994.

[8] T. Bräuner, An Axiomatic Approach to Adequacy, Ph.D. Dissertation, BRICS Dissertation Series DS-96-4, University of Aarhus, 1996.

[9] J.D. Brock, W.B. Ackerman, Scenarios: a model of non-determinate computation, in: Proc. Formalization of Programming Concepts, Lecture Notes in Computer Science, Vol. 107, Springer, Berlin, 1981, pp. 252–259.

[10] L. Cardelli, A.D. Gordon, Anytime, anywhere: modal logics for mobile ambients, in: Proc. POPL'00.

[11] L. Cardelli, A.D. Gordon, A commitment relation for the ambient calculus, October 6th, 2000, available from http://research.microsoft.com/~adg/.

[12] G.L. Cattani, Presheaf Models for Concurrency, Ph.D. Dissertation, BRICS Dissertation Series DS-99-1, University of Aarhus, 1999.

[13] G.L. Cattani, I. Stark, G. Winskel, Presheaf models for the π-calculus, in: Proc. CTCS'97, Lecture Notes in Computer Science, Vol. 1290, Springer, Berlin, 1997, pp. 106–126.

[14] G.L. Cattani, G. Winskel, Presheaf models for concurrency, in: Proc. CSL'96, Lecture Notes in Computer Science, Vol. 1258, Springer, Berlin, 1996, pp. 58–75.

[15] G.L. Cattani, G. Winskel, Profunctors, open maps and bisimulation, Manuscript, 2003. Available from http://www.cl.cam.ac.uk/~gw104/.

[16] F. Crazzolara, G. Winskel, Events in security protocols, in: Proc. 8th ACM Conference on Computer and Communication Security, 2001.

[17] J.B. Dennis, Data flow computation, in: M. Broy (Ed.), Control Flow and Data Flow: Concepts of Distributed Programming, Springer, Berlin, 1985.

[18] S. Furber (Ed.), Proc. Eighth International Symposium on Asynchronus Circuits and Systems, IEEE Press, New York, 2002.

[19] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1) (1987) 1–102.

[20] M. Hennessy, A fully abstract denotational model for higher-order processes, Inform. and Comput. 112 (1) (1994) 55–95.

[21] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, J. ACM 32 (1) (1985) 137–161.

[22] M.C.B. Hennessy, G.D. Plotkin, Full abstraction for a simple parallel programming language, in: Proc. MFCS'79, Lecture Notes in Computer Science, Vol. 74, Springer, Berlin, 1979, pp. 108–120.

[23] T. Hildebrandt, P. Panangaden, G. Winskel, A relational model of non-deterministic dataflow, in: Proc. CONCUR'98, Lecture Notes in Computer Science, Vol. 1466, Springer, Berlin, 1998, pp. 613–628.

[24] C.A.R. Hoare, Some properties of predicate transformers, J. ACM 25 (3) (1987) 461–480.

[25] B. Jacobs, Semantics of weakening and contraction, Ann. Pure Appl. Logic 69 (1) (1994) 73–106.

[26] A. Joyal, M. Nielsen, G. Winskel, Bisimulation from open maps, Inform. and Comput. 127 (1996) 164–185.

[27] G. Kahn, The semantics of a simple language for parallel programming, in: J.L. Rosenfeld (Ed.), Information Processing 74, North-Holland, Amsterdam, 1974, pp. 471–475.

[28] G.M. Kelly, Doctrinal adjunction, in: Proc. Category Seminar, Sydney, Lecture Notes in Mathematics, Vol. 420, Springer, Berlin, 1972/73.
[29] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Comm. ACM 21 (7) (1978) 558–565.
[30] K.G. Larsen, G. Winskel, Using information systems to solve recursive domain equations effectively, in: Proc. Semantics of Data Types, Lecture Notes in Computer Science, Vol. 173, Springer, Berlin, 1984, pp. 109–129.
[31] S. Mac Lane, Categories for the Working Mathematician, 2nd Edition, Springer, Berlin, 1998.
[32] G. McCusker, A fully abstract relational model of syntactic control of interference, in: Proc. CSL'02, Lecture Notes in Computer Science, Vol. 2471, Springer, Berlin, 2002, pp. 247–261.
[33] P.-A. Melliès, Categorical models of linear logic revisited, Prépublication de l'Equipe PPS no. 22, Université Denis Diderot, 2003; Theoret. Comput. Sci. to appear.
[34] R. Milner, A Calculus of Communicating Systems, in: Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1980.
[35] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[36] R. Milner, Communicating and Mobile Systems: the π-Calculus, Cambridge University Press, Cambridge, 1999.
[37] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, parts I and II, Inform. and Comput. 100 (1) (1992) 1–77.
[38] J.H. Morris, Lambda-calculus models of programming languages, Ph.D. Thesis, MIT, Cambridge, MA, 1968.
[39] M. Nielsen, G. Plotkin, G. Winskel, Petri nets, event structures and domains, part I, Theoret. Comput. Sci. 13 (1) (1981) 85–108.
[40] M. Nygaard, Domain theory for concurrency, Ph.D. Dissertation, University of Aarhus, 2003, available from http://www.daimi.au.dk/~nygaard/.
[41] M. Nygaard, G. Winskel, Linearity in process languages, in: Proc. LICS'02, 2002, pp. 433–446.
[42] M. Nygaard, G. Winskel, HOPLA—a higher-order process language, in: Proc. CONCUR'02, Lecture Notes in Computer Science, Vol. 2421, Springer, Berlin, 2002, pp. 434–448.
[43] M. Nygaard, G. Winskel, Full abstraction for HOPLA, in: Proc. CONCUR'03, Lecture Notes in Computer Science, Vol. 2761, Springer, Berlin, 2003, pp. 378–392.
[44] D. Park, Concurrency and automata on infinite sequences, in: Proc. Theoretical Computer Science: 5th GI-Conference, Lecture Notes in Computer Science, Vol. 104, 1981, pp. 167–183.
[45] D.A. Peled, V.R. Pratt, G.J. Holzmann (Eds.), Proc. Partial Order Methods in Verification 1996, DIMACS 29, American Mathematical Society, Providence, RI, 1997.
[46] G. Plotkin, A powerdomain construction, SIAM J. Comput. 5 (3) (1976) 452–487.
[47] D. Sangiorgi, D. Walker, The π-calculus. A Theory of Mobile Processes, Cambridge University Press, Cambridge, 2001.
[48] D.S. Scott, Domains for denotational semantics, in: Proc. ICALP'82, Lecture Notes in Computer Science, Vol. 140, Springer, Berlin, 1982, pp. 577–613.
[49] R.A.G. Seely, Linear logic, $*$-autonomous categories and cofree coalgebras, in: Proc. Categories in Computer Science and Logic, 1987, Contemporary Mathematics, Vol. 92, American Mathematical Society, Providence, RI, 1987.
[50] F.J. Thayer, J.C. Herzog, J.D. Guttman, Strand spaces: why is a security protocol correct? in: Proc. IEEE Symp. on Security and Privacy, 1998.
[51] B. Thomsen, A calculus of higher-order communicating systems, in: Proc. POPL'89, 1989, pp. 143–154.
[52] G. Winskel, On powerdomains and modality, Theoret. Comput. Sci. 36 (1985) 127–137.
[53] G. Winskel, The Formal Semantics of Programming Languages. An Introduction, The MIT Press, Cambridge, MA, 1993.
[54] F.Z. Nardelli, De la Sémantique des processus d'ordre supérieur, Ph.D. Thesis, Université Paris 7, 2003.
[55] G. Winskel, M. Nielsen, Models for concurrency, in: S. Abramsky, et al. (Eds.), Handbook of Logic in Computer Science, Semantic Modelling, Vol. 4, Oxford University Press, Oxford, 1995.